Proceedings of the 2nd Workshop on

New Frontiers in Mining Complex Patterns (NFMCP 2013)

Editors

Annalisa Appice

Michelangelo Ceci

Corrado Loglisci

Giuseppe Manco

Elio Masciari

Zbigniew Ras

New Frontiers in Mining Complex Patterns (NFMCP 2013)

The second International Workshop on New Frontiers in Mining Complex Patterns (NFMCP 2013) was held in Prague CZ, on September 27th 2013 in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2013).

This workshop starts from awareness that Data mining and Knowledge discovery can be considered today as mature research fields with numerous algorithms and studies to extract knowledge from data in different forms. Although most existing data mining approaches look for patterns in tabular data, there are also numerous studies which look for patterns in complex data (e.g. multi-table data, XML data, web data, time series and sequences, graphs and trees). The recent developments in technologies and life sciences have paved the way to the proliferation of data collections representing new complex interactions between entities in distributed and heterogeneous sources. These interactions may be spanned at multiple levels of granularity as well as at spatial and temporal dimensions.

The workshop is not the first of its kind. It follows the successful first edition (NFMCP 2012) and the national workshop on Mining Complex Patterns which was held in conjunction with Italian Conference on Artificial Intelligence AI*IA 2011 as well as several editions of the international workshop on Mining Complex Data (MCD 2006@IEEE ICDM 2006, MCD 2007@ECML/PKDD 2007, MCD 2008 @ IEEE ICDM 2008).

Our purpose in this workshop was to bring together researchers and practitioners in the data mining area interested in exploring emerging technologies and applications where complex patterns in expressive languages are principally extracted from new prominent data sources like blogs, event or log data, biological sequence data, spatio-temporal data, social networks, mobility data, sensor data and streams, and so on. We were interested in advanced techniques which preserve the informative richness of data and allow us to efficiently and efficaciously identify complex information units present in such data.

We received twenty-four submissions in several research fields ranging from multi-relational data mining to spatio-temporal data mining, stream data mining, graph and network mining, process mining, bio-medic and music mining. We were able to accept twenty-one papers, based on a rigorous reviewing process. Each submission was evaluated by three independent referees. Additionally, the scientific program also featured an invited talk on "Evolving Social Networks: trajectories of communities" by João Gama (IAAD-INESC TEC and Faculty of Economics, University of Porto, Porto, Portugal).

We would like to thank the invited speaker, all the authors who submitted papers and all the workshop participants. We are also grateful to the members of the program committee and external referees for their thorough work in re-

viewing submitted contributions with expertise and patience. A special thank is due to both the ECML PKDD Workshop Chairs and to the members of ECML PKDD organizers who made this event possible.

Annalisa Appice,
Michelangelo Ceci,
Corrado Loglisci,
Giuseppe Manco,
Elio Masciari,
Zbigniew Ras.

September 2013.

Organization

Program Chairs

Annalisa Appice	University of Bari "Aldo Moro", Bari, Italy
Michelangelo Ceci	University of Bari "Aldo Moro", Bari, Italy
Corrado Loglisci	University of Bari "Aldo Moro", Bari, Italy
Giuseppe Manco	ICAR-CNR, Rende, Italy
Elio Masciari	ICAR-CNR, Rende, Italy
Zbigniew Ras	University of North Carolina, Charlotte, USA & Warsaw University of Technology, Poland

Program Committee

Nicola Barbieri (Yahoo Research, Spain)
Petr Berka (University of Economics Prague)
Sašo Džeroski (Jozef Stefan Institute)
Floriana Esposito (University of Bari "Aldo Moro")
Dimitrios Gunopulos (University of Athens)
Mohand-Saïd Hacid (University Claude Bernard Lyon 1)
Dino Ienco (IRSTEA Montpellier, UMR TETIS)
Kristian Kersting (Fraunhofer IAIS, Sankt Augustin)
Arno Knobbe (University of Leiden)
Stan Matwin (University of Ottawa)
Dino Pedreschi (University of Pisa)
Jean-Marc Petit (INSA-Lyon, LIRIS)
Fabrizio Riguzzi (University of Ferrara)
Henryk Rybiński (Warsaw University of Technology)
Eirini Spyropoulou (University Of Bristol)
Jerzy Stefanowski (Poznan University of Technology)
Maguelonne Teisseire (IRSTEA Montpellier, UMR TETIS)
Herna Viktor (University of Ottawa)
Alicja Wieczorkowska (Polish-Japanese Institute of IT)
Wlodek Zadrozny (IBM Watson Research Center)
Djamel Zighed (Université Lumière Lyon 2)

Additional Reviewers

Stefano Ferilli
Matteo Riondato

Riccardo Ortale
Massimo Guarascio
Pance Panov
Nhat Hai Phan

Table of Contents

Invited Talk

Contribution Papers

Evolving Social Networks: trajectories of communities

João Gama

LIAAD-INESC TEC and Faculty of Economics, University of Porto, Porto, Portugal

Abstract. In recent years we witnessed an impressive advance in the social networks field, which became a "hot" topic and a focus of considerable attention. The development of methods that focus on the analysis and understanding of the evolution of data are gaining momentum. The need for describing and understanding the behavior of a given phenomenon over time led to the emergence of new frameworks and methods focused in temporal evolution of data and models. In this talk we discuss the research opportunities opened in analysing evolving data and present examples from mining the evolution of clusters and communities in social networks.

Structure Determination and Estimation of Hierarchical Archimedean Copulas Based on Kendall Correlation Matrix

Jan Górecki[1], Martin Holeňa[2]

[1] Department of Informatics
SBA in Karvina, Silesian University in Opava
Karvina, Czech Republic
gorecki@opf.slu.cz
[2] Institute of Computer Science
Academy of Sciences of the Czech Republic
Praha, Czech Republic
martin@cs.cas.cz

Abstract. Copulas recently emerged in many data analysis and knowledge discovery tasks as a flexible tool for modeling complex multivariate distributions. The paper presents a method for estimating copulas from one of the most popular classes of copulas, namely hierarchical Archimedean copulas. The method is based on the close relationship of the copula structure and the values of Kendall's tau computed on all its bivariate margins. A simple algorithm implementing the method is provided and its effectiveness is shown in several experiments including its comparison to other available methods.

Keywords: hierarchical Archimedean copula, estimation, structure determination, Kendall's correlation coefficient

1 Introduction

Despite the fact that copulas have most success in finance, they are increasingly adopted by researchers from many other application areas. We can see applications of copulas in water-resources and hydro-climatic analysis [1, 8], gene analysis [10], cluster analysis [9, 16] or in evolution algorithms, particularly in the estimation of distribution algorithms [3, 20]. In the applications that can be generally put in the framework of knowledge discovery and data mining, copulas are used due to their effective mathematical ability to capture complex dependence structures among variables. For illustrative example, we refer to [8], where the task for anomaly detection in climate that incorporates complex spatio-temporal dependencies is solved using copulas.

Hierarchical Archimedean copulas (HACs) are a frequently used alternative to the most popular Gaussian copula due to their flexibility and conveniently limited number of parameters. Despite their popularity, feasible techniques for

HAC estimation are addressed only in few papers. Most of them assume in the estimation process a given structure of a copula, which is motivated trough applications in economy, see [17, 18]. There exists only one recently published paper, see [13], which address the estimation technique generally, i.e., the estimation also concerns the proper structure determination of a HAC.

The mentioned paper describes a multi-stage procedure, which is used both for the structure determination and the estimation of the parameters. The authors devote mainly to the estimation of the parameters using the maximum-likelihood (ML) technique and briefly mention its alternative, which uses for the parameters estimation the relationship between the copula parameter and the value of Kendall's tau computed on a bivariate margin of the copula (shortly, $\theta-\tau$ relationship). The authors present six approaches denoted as $\tau_{\Delta\tau>0}, \tau_{binary}$, Chen, θ_{binary}, $\theta_{binary\ aggr.}$ and θ_{RML} to the structure determination based on the both mentioned estimation techniques (the ML and the $\theta-\tau$ relationship). The first five of them lead to biased estimators, what can be seen in the results of the attached simulation study, and the the sixth (θ_{RML}) is used for re-estimation and thus for better approximation of the parameters of the true copula. θ_{RML} shows the best goodness-of-fit (measured by Kullback-Leibler divergence) of the resulting estimates. However, the best approximation of the true parameters with θ_{RML} is possible only in the cases, when the structure is properly determined (the estimated structure equals the true structure). But, as θ_{RML} is based on the biased $\theta_{binary\ aggr.}$, which often does not return the true structure due to the involved bias, θ_{RML} also cannot return close approximation of the true parameters in the cases, when the structure is determined improperly. Moreover, the number of those cases rapidly increases with the increasing data dimension, as we show later in Section 4.

In our paper we propose the construction of an estimator for HACs, which approximates the parameters of the true copula better than the previously mentioned methods, and thus also increases the ratio of properly determined structures. Avoiding the need of re-estimation, we also gain high computational efficiency. The included experiments on simulated data show that our approach outperforms all the other above mentioned methods in the sense of goodness-of-fit, the properly determined structures ratio and also in the time consumption, which is even slightly lower than the most efficient binary methods $\tau_{binary}, \theta_{binary}$.

The paper is structured as follows. The next section summarizes some necessary theoretical concepts concerning Archimedean copulas (ACs) and HACs. Section 3 presents the new approach to the HAC estimation. Section 4 describes the experiments and their results and Section 5 concludes this paper.

2 Preliminaries

2.1 Copulas

Definition 1. *For every $d \geq 2$, a* d-dimensional copula *(shortly,* d-copula*) is a d-variate distribution function on $\mathbb{I}^d$ ($\mathbb{I}$ is the unit interval), whose univariate margins are uniformly distributed on $\mathbb{I}$.*

Theorem 1. ***(Sklar's Theorem)*** *[19] Let H be a d-dimensional d.f. with univariate margins $F_1, ..., F_d$. Let A_j denote the range of F_j, $A_j := F_j(\overline{\mathbb{R}})(j = 1, ..., d), \overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$. Then there exists a copula C such for all $(x_1, ..., x_d) \in \overline{\mathbb{R}}^d$,*

$$H(x_1, ..., x_d) = C(F_1(x_1), ..., F_d(x_d)). \tag{1}$$

Such a C is uniquely determined on $A_1 \times ... \times A_d$ and, hence, it is unique if $F_1, ..., F_d$ are all continuous. Conversely, if $F_1, ..., F_d$ are univariate d.f.s, and if C is any d-copula, then the function $H : \overline{\mathbb{R}}^d \to \mathbb{I}$ defined by (1) is a d-dimensional distribution function with margins $F_1, ..., F_d$.

Through the Sklar's theorem, one can derive for any d-variate d.f. its copula C using (1). In case that the margins $F_1, ..., F_d$ are all continuous, the copula C is given by $C(u_1, ..., u_d) = H(F_1^-(u_1), ..., F_d^-(u_d))$, where $F_i^-, i \in \{1, ..., d\}$ denotes pseudo-inverse of F_i given by $F_i^-(s) = \inf\{t|\ F_i(t) \geq s\}, s \in \mathbb{I}$. Many classes of copulas are derivable in this way from popular joint d.f.s, e.g., the most popular class of Gaussian copulas is derived using H corresponding to a d-variate Gaussian distribution. But, using this process often results in copulas not expressible in closed form, what can bring difficulties in some applications.

2.2 Archimedean Copulas

This drawback is overcame while using Archimedean copulas, due to their different construction process. ACs are not constructed using the Sklar's theorem, but instead of it, one starts with a given functional form and asks for properties needed to obtain a proper copula. As a result of such a construction, ACs are always expressed in closed form, which is one of the main advantages of this class of copulas [6]. To construct ACs, we need the notion of an *Archimedean generator* and of a *complete monotonicity*.

Definition 2. Archimedean generator *(shortly,* generator*) is continuous, non-increasing function $\psi : [0, \infty] \to [0, 1]$, which satisfies $\psi(0) = 1, \psi(\infty) = \lim_{t\to\infty} \psi(t) = 0$ and is strictly decreasing on $[0, \inf\{t : \psi(t) = 0\}]$. We denote the set of all generators as Ψ.*

Definition 3. *A function f is called* completely monotone *(shortly, c.m.) on $[a, b]$, if $(-1)^k f^{(k)}(x) \geq 0$ holds for every $k \in \mathbb{N}_0, x \in (a, b)$.*

Definition 4. *Any d-copula C is called* Archimedean copula *(we denote it d-AC), if it admits the form*

$$C(\mathbf{u}) := C(\mathbf{u}; \psi) := \psi(\psi^{-1}(u_1) + ... + \psi^{-1}(u_d)), \boldsymbol{u} \in \mathbb{I}^d, \tag{2}$$

where $\psi \in \Psi$ and its inverse $\psi^{-1} : [0, 1] \to [0, \infty]$ is defined $\psi^{-1}(s) = \inf\{t : \psi(t) = s\}, s \in \mathbb{I}$.

For verifying whether function C given by (2) is a proper copula, we can use the property stated in Definiton 3. A condition sufficient for C to be a copula is stated as follows.

Theorem 2. *[11] If $\psi \in \Psi$ is completely monotone, then the function C given by (2) is a copula.*

We can see from Definition 4 and the properties of generators that having a random vector $\mathbf{U}$ distributed according to some AC, all its k-dimensional ($k < d$) marginal copulas have the same marginal distribution. It implies that all multivariate margins of the same dimension are equal, thus, e.g., the dependence among all pairs of components is identical. This symmetry of ACs is often considered to be a rather strong restriction, especially in high dimensional applications.

Given the number of variables, to derive the explicit form of an AC to work with, we need the explicit form of generators. The reader can find many explicit forms of the generators in, e.g., [12]. In this paper, we use and present only the Clayton generator, defined $(1+t)^{-1/\theta}$. Copulas based on this generator have been used, e.g., to study correlated risks, because they exhibit strong left tail dependence and relatively weak right tail dependence. The explicit parametric form of a bivariate Clayton copula is $C(u_1, u_2; \psi) = \left({u_1}^{-\theta} + {u_2}^{-\theta} - 1\right)^{-\frac{1}{\theta}}$. [12]

2.3 Hierarchical Archimedean Copulas

To allow for asymmetries, one may consider the class of HACs (often also called *nested Archimedean copulas*), recursively defined as follows.

Definition 5. *[7] A d-dimensional copula C is called* hierarchical Archimedean copula *if it is an AC with arguments possibly replaced by other hierarchical Archimedean copulas. If C is given recursively by (2) for $d = 2$ and*

$$C(\mathbf{u}; \psi_0, ..., \psi_{d-2}) = \psi_0(\psi_0^{-1}(u_1) + \psi_0^{-1}(C(u_2, ..., u_d; \psi_1, ..., \psi_{d-2}))), \boldsymbol{u} \in \mathbb{I}^d, \quad (3)$$

for $d \geq 3$, C is called fully-nested hierarchical Archimedean copula *with $d-1$* nesting levels. *Otherwise C is called* partilally-nested hierarchical Archimedean copula.

Remark 1. We denote a d-dimensional HAC as d-HAC. For some d-HAC, we refer to the hierarchical ordering of all ACs $C(\cdot; \psi_0), ..., C(\cdot; \psi_k), k \leq d-2$ incorporated in the d-HAC together with the ordering of variables $u_1, ..., u_d$ as the *structure* of the d-HAC.

From the definition, we can see that ACs are special cases of HACs. The most simple proper fully-nested HAC is obtained for $d = 3$ and is with two nesting levels. The structure of this copula is given by

$$\begin{aligned} C(\mathbf{u}; \psi_0, \psi_1) &= C(u_1, C(u_2, u_3; \psi_1); \psi_0) \\ &= \psi_0(\psi_0^{-1}(u_1) + \psi_0^{-1}(\psi_1(\psi_1^{-1}(u_2) + \psi_1^{-1}(u_3)))), \mathbf{u} \in \mathbb{I}^3. \end{aligned} \quad (4)$$

As in the case of ACs, we can ask for necessary and sufficient condition for the function C given by (3) to be a proper copula. Partial answer for this question in form of sufficient condition is contained in the following theorem.

Theorem 3. ***(McNeil (2009))*** *[11] If $\psi_j \in \Psi_\infty, j \in \{0, ..., d-2\}$ such that $\psi_k^{-1} \circ \psi_{k+1}$ have completely monotone derivatives for all $k \in \{0, ..., d-3\}$, then $C(\boldsymbol{u}; \psi_0, ..., \psi_{d-2})$, $\boldsymbol{u} \in \mathbb{I}^d$, given by (3) is a copula.*

McNeil's theorem is stated only for fully-nested HACs, but it can be easily translated also for use with partially-nested HACs (for more see [6]). The condition for $(\psi_0^{-1} \circ \psi_1)'$ to be compete monotone is often called the *nesting condition*.

When using HACs in applications, there exist, for example for $d = 10$, more than 280 millions of possible HAC structures and each 10-HAC can incorporate up to 9 parameters (using only one-parametric generators) in generators from possibly different families. If choosing the model that the best fit the data, this is much more complex situation relative to the case when using ACs, which have just one structure, one parameter and one Archimedean family.

For the sake of simplicity, assume that each d-HAC structure corresponds to some binary tree t. Each node in t represents one 2-AC. Each 2-AC is determined just by its corresponding generator, so we identify each node in t with one generator and hence we have always nodes $\psi_0, ..., \psi_{d-2}$. For a node ψ denote as $\mathcal{D}_n(\psi)$ the set of all descendant nodes of ψ, $\mathcal{P}(\psi)$ the parent node of ψ, $\mathcal{H}_l(\psi)$ the left child of ψ and $\mathcal{H}_r(\psi)$ the right child of ψ. The leafs of t correspond to the variables $u_1, ..., u_d$.

2.4 Kendall's tau and its generalization

As we are interested in Kendall's tau relationship with a general bivariate copula, we use its definition given by (as in [1])

$$\tau(C) = 4 \int_{\mathbb{I}^2} C(u_1, u_2) dC(u_1, u_2) - 1. \tag{5}$$

If C is a 2-AC based on a generator ψ, and ψ depends on the parameter $\theta \in \mathbb{R}$, then (5) states an explicit relationship between θ and τ, which can be often expressed in a closed form. For example, if C is a Clayton copula, we get $\tau = \theta/(\theta+2)$ (the relationship between θ and τ for other generators can be found, e.g., in[6]). The inversion of this relationship establish an estimator of the parameter θ, which can be based on the empirical version of τ given by (as in [1])

$$\tau_n = \frac{4}{n(n-1)} \sum_{i=1,j=1}^{n} \mathbf{1}_{\{(u_{i1}-u_{j1})(u_{i2}-u_{j2})>0\}}, \tag{6}$$

where $(u_{\bullet 1}, u_{\bullet 2})$ denotes the realizations of r.v.s $(U_1, U_2) \sim C$.

To generalize τ for m (possibly > 2) random variables (r.v.s) we state the following definition. For simplification denote the set of pairs of r.v.s as $\mathbf{U}_{IJ} = \{(U_i, U_j) | (i, j) \in I \times J\}$, where $I, J \subset \{1, ..., d\}, I \neq \emptyset \neq J, (U_1, ..., U_d) \sim C$, C is a d-HAC.

Definition 6. *Let τ be the Kendall's tau, g be an aggregation function (like, e.g., max, min or mean), which has the following properties: 1) $g(u, ..., u) = u$*

for all $u \in \mathbb{I}$ *and 2)* $g(u_{p_1}, ..., u_{p_k}) = g(u_1, ..., u_k)$ *for all* $u_1, ..., u_k \in \mathbb{I}$ *and all permutations* p *of* $\{1, ..., k\}$*. Then define an* aggregated *Kendall's tau* τ^g *as*

$$\tau^g(\mathbf{U}_{IJ}) = \begin{cases} \tau(U_i, U_j) & \text{if } I = \{i\}, J = \{j\} \\ g(\tau(U_{i_1}, U_{j_1}), \tau(U_{i_1}, U_{j_2}), ..., \tau(U_{i_l}, U_{j_q})), & \text{else,} \end{cases} \quad (7)$$

where $I = \{i_1, ..., i_l\}, J = \{j_1, ..., j_q\}, i_l, j_q \leq d$ *are non-empty disjoint subsets of* $\mathbb{N}$*.*

2.5 Okhrin's algorithm for the structure determination of HAC

We recall the algorithm presented in [14] for the structure determination of HAC, which returns for some unknown HAC C its structure using only the known forms of its bivariate margins. The algorithm uses the following definition.

Definition 7. *Let* C *be a* d*-HAC with generators* $\psi_0, ..., \psi_{d-2}$ *and* $(U_1, ..., U_d) \sim C$*. Then denote as* $\mathcal{U}_C(\psi_k), k = 0, ..., d-2$*, the set of indexes* $\mathcal{U}_C(\psi_k) = \{i | (\exists U_j)(U_i, U_j) \sim C(\cdot; \psi_k) \vee (U_j, U_i) \sim C(\cdot; \psi_k), 1 \leq i < j \leq d\}, k = 0, ..., d-2$*.*

Proposition 1. *[14] Defining* $\mathcal{U}_C(u_i) = \{i\}$ *for the leaf* $i, 1 \leq i \leq d$*, there is an unique disjunctive decomposition of* $\mathcal{U}_C(\psi_k)$ *given by*

$$\mathcal{U}_C(\psi_k) = \mathcal{U}_C(\mathcal{H}_l(\psi_k)) \cup \mathcal{U}_C(\mathcal{H}_r(\psi_k)). \quad (8)$$

For an unknown d-HAC C, knowing all its bivariate margins, its structure can be easily determined with Algorithm 1, which returns the unknown structure t of C. We start from the sets $\mathcal{U}_C(u_1), ..., \mathcal{U}_C(u_d)$ joining them together through (8) until we reach the node ψ for which $\mathcal{U}_C(\psi) = \{1, ..., d\}$.

Algorithm 1 The HAC structure determination

Input: 1) $\mathcal{U}_C(\psi_0), ..., \mathcal{U}_C(\psi_{d-2})$, 2) $\mathcal{I} = \{0, ..., d-2\}$
while $\mathcal{I} \neq \emptyset$ **do**
1. $k = \text{argmin}_{i \in \mathcal{I}}(\#\mathcal{U}_C(\psi_i))$, if there are more minima, then choose as k one of them arbitrarily.
2. Find the nodes ψ_l, ψ_r, for which $\mathcal{U}_C(\psi_k) = \mathcal{U}_C(\psi_l) \cup \mathcal{U}_C(\psi_r)$.
3. $\mathcal{H}_l(\psi_k) := \psi_l, \mathcal{H}_r(\psi_k) := \psi_r$.
4. Set $\mathcal{I} := \mathcal{I} \backslash \{k\}$.
end while
Output: The structure stored in $\mathcal{H}_l(\psi_k), \mathcal{H}_r(\psi_k), k = 0, ..., d-2$

3 Our Approach

3.1 HAC structure determination

Recalling Theorem 3, the sufficient condition for C to be a proper copula is that the nesting condition must hold for each generator and its parent in a HAC

structure. As this is the only known condition that assures that C is a proper copula, we deal in out work only the copulas, which fulfill this condition. The nesting condition results in constraints on the parameters θ_0, θ_1 of the involved generators ψ_0, ψ_1 (see [6, 7]). As $\theta_i, i = 1, 2$ is closely related to τ through (5), there is also an important relationship between the values of τ and the HAC tree structure following from the nesting condition. This relationship is described for the fully-nested 3-HAC given by the form (4) in Remark 2.3.2 in [6]. There, it is shown that if the nesting condition holds for the parent-child pair (ψ_0, ψ_1), then $0 \leq \tau(\psi_0) \leq \tau(\psi_1)$ (as we deal only with HACs with binary structures incorporating only 2-ACs, which are fully determined only by its generator, we use as the domain of τ the set Ψ instead of the usually used set of all 2-copulas). We generalize this statement, using our notation, as follows.

Proposition 2. *Let C be a d-HAC with the structure t and the generators $\psi_0, ..., \psi_{d-2}$, where each parent-child pair satisfy the nesting condition. Then $\tau(\psi_i) \leq \tau(\psi_j)$, where $\psi_j \in \mathcal{D}_n(\psi_i)$, holds for each $\psi_i, i = 0, ..., d-2$.*

Proof. As $\psi_j \in \mathcal{D}(\psi_i)$, there exists a unique sequence $\psi_{k_1}, ..., \psi_{k_l}$, where $0 \leq k_m \leq d-2, m = 1, ..., l, l \leq d-1, \psi_{k_1} = \psi_i, \psi_{k_l} = \psi_j$ and $\psi_{k-1} = \mathcal{P}(\psi_k)$ for $k = 2, ..., l$. Applying the above mentioned remark for each pair $(\psi_{k-1}, \psi_k), k = 2, ..., l$, we get $\tau(\psi_{k_1}) \leq ... \leq \tau(\psi_{k_l})$. □

Thus, having a branch from t, all its nodes are uniquely ordered according to their value of τ assuming unequal values of τ for all parent-child pairs. This provides an alternative algorithm for the HAC structure determination. We have to assign the generators with the highest values of τ to the lowest levels of the branches in the structure and ascending to higher levels we assign the generators with lower values of τ.

Remark 2. $\tau(\psi_k) = \tau^g(\mathbf{U}_{\mathcal{U}_C(\mathcal{H}_l(\psi_k))\mathcal{U}_C(\mathcal{H}_r(\psi_k))})$ for a d-HAC C and for each $k = 0, ..., d-2$. This is because the bivariate margins $C_{ij}, (i, j) \in \mathcal{U}_C(\mathcal{H}_l(\psi_k)) \times \mathcal{U}_C(\mathcal{H}_r(\psi_k))$ of C are all equal and $g(u, ..., u) = u$ for all $u \in \mathbb{I}$. Thus $\tau(\psi_k)$ depends only on the population version of Kendall correlation matrix.

Computing $\tau(\psi_k), k = 0, ..., d-2$ using Remark 2 and following Proposition 2 leads to the alternative algorithm for HAC structure determination. The algorithm is summarized in Algorithm 2 and can be used for arbitrary $d > 2$ (see [4] for more details including an example for $d = 4$). It returns the sets $\mathcal{U}_C(z_{d+k+1})$ corresponding to the sets $\mathcal{U}_C(\psi_k), k = 0, ..., d-2$. Passing them to Algorithm 1, we avoid their computation from Definition 7 and we get the requested d-HAC structure without a need of knowing the forms of the bivariate margins. Assuming a family for each ψ_k, $\theta - \tau$ relationship for the given family can be used to obtain the parameters, i.e., $\theta_k = \tau_\theta^{-1}(\tau(\psi_k)), k = 0, ..., d-2$, where τ_θ^{-1} denotes the $\theta - \tau$ relationship, e.g., for Clayton family $\tau_\theta^{-1}(\tau) = 2\tau/(1-\tau)$. Hence we get together with the structure the whole copula.

Algorithm 2 The HAC structure determination based on κ

Input: 1) $\mathcal{I} = \{1, ..., d\}$, 2) $(U_1, ..., U_d) \sim C$, 3) τ^g ... an aggregated Kendall's tau, 4) $z_k = u_k, \mathcal{U}_C(z_k) = \{k\}, k = 1, ..., d$

The structure determination:

for $k = 0, ..., d-2$ **do**

1. $(i, j) := \underset{i^*<j^*, i^* \in \mathcal{I}, j^* \in \mathcal{I}}{\operatorname{argmax}} \tau^g(\mathbf{U}_{\mathcal{U}_C(z_{i^*})\mathcal{U}_C(z_{j^*})})$
2. $\mathcal{U}_C(z_{d+k+1}) := \mathcal{U}_C(z_i) \cup \mathcal{U}_C(z_j)$
3. $\mathcal{I} := \mathcal{I} \cup \{d+k+1\} \backslash \{i, j\}$

end for

Output: $\mathcal{U}_C(z_{d+k+1}), k = 0, ..., d-2$

3.2 HAC estimation

As the aggregated τ^g depends only on the pairwise τ and the aggregation function g, we can easily derive its empirical version τ_n^g just by substituting τ in τ^g by its empirical version τ_n given by (6). Using τ_n^g instead of τ^g we can easily derive the empirical version of the structure determination process represented by Algorithms 1, 2.

In this way we base the structure determination only on the values of the pairwise τ. This is an essential property of our approach. Using the $\theta - \tau$ relationship established through (5) for some selected Archimedean family, whole HAC, including its structure and its parameters, can be estimated just from Kendall correlation matrix computed for the realizations of $(U_1, ..., U_d)$ assuming all the generators to be from the selected Archimedean family. Due to nesting condition, the parameter $\hat{\theta}_k$ is trimmed in Step 3. in order to obtain the resulting estimate as a proper d-HAC. Note that if we allow the generators to be from different Archimedean families, the task is much more complex, and we do not concern it in the paper due to space limitations and refer the reader to [5, 6].

The proposed empirical approach is summarized in Algorithm 3. The Kendall correlation matrix (τ_{ij}^n) is computed for the realizations of the pairs $(U_i, U_j), 1 \leq i < j \leq d$ using (6). The algorithm returns the parameters $\hat{\theta}_0, ..., \hat{\theta}_{d-2}$ of the estimate $\hat{C}$ and the sets $\mathcal{U}_{\hat{C}}(z_{d+k+1})$ corresponding to the sets $\mathcal{U}_{\hat{C}}(\psi_k), k = 0, ..., d-2$. Passing the sets to Algorithm 1 we get the requested $\hat{C}$ structure.

4 Experiments

We performed a lot of different experiments on simulated data involving different data dimensions, HAC structures, generators and parameters. Due to space limitations we present only one experiment, where we compare the proposed method with the other previously mentioned methods on simulated data for $d = 5, 6, 7, 9$. We simulate 100 samples of size 500 according to [7] for 4 copula models based on the Clayton generator. The first considered model is $((12)_{\frac{3}{4}}(3(45)_{\frac{4}{4}})_{\frac{3}{4}})_{\frac{2}{4}}$. The natural numbers in the model notation (as in [13]) are the indexes of the copula variables, i.e., 1,...5, the parentheses correspond to each $\mathcal{U}_C(\cdot)$, i.e.,

Algorithm 3 The HAC estimation

Input: 1) (τ_{ij}^n) {...Kendall correlations matrix}, 2) g {...an aggregation function}, 3) $\mathcal{I} = \{1, ..., d\}$, 4) $z_i = u_i, i = 1, ..., d$, 5) Archimedean family and corresponding τ_θ^{-1}

Estimation:

for $k = 0, ..., d-2$ **do**

1. $(i, j) := \underset{\tilde{i}<\tilde{j}, \tilde{i}\in\mathcal{I}, \tilde{j}\in\mathcal{I}}{\operatorname{argmax}} g((\tau^n_{\tilde{z}_{\tilde{i}}\tilde{z}_{\tilde{j}}})_{(\tilde{i},\tilde{j})\in\mathcal{U}_{\hat{C}}(z_{\tilde{i}})\times\mathcal{U}_{\hat{C}}(z_{\tilde{j}})})$
2. $\hat{\theta}_k := \tau_\theta^{-1}\big(g((\tau^n_{\tilde{i}\tilde{j}})_{(\tilde{i},\tilde{j})\in\mathcal{U}_{\hat{C}}(z_i)\times\mathcal{U}_{\hat{C}}(z_j)})\big)$
3. $\hat{\theta}_k := \min(\hat{\theta}_k, \hat{\theta}_i, \hat{\theta}_j)$
4. $\mathcal{U}_{\hat{C}}(z_{d+k+1}) := \mathcal{U}_{\hat{C}}(i) \cup \mathcal{U}_{\hat{C}}(j)$
5. $\mathcal{I} := \mathcal{I} \cup \{d+k+1\} \backslash \{i, j\}$

end for

Output: $\hat{\theta}_k, \mathcal{U}_{\hat{C}}(k), k = 0, ..., d-2$

$\mathcal{U}_C(\psi_0) = \{1, 2\}$,$\mathcal{U}_C(\psi_1) = \{4, 5\}$,$\mathcal{U}_C(\psi_2) = \{3, 4, 5\}$,$\mathcal{U}_C(\psi_3) = \{1, 2, 3, 4, 5\}$, and the subscripts are the model parameters, i.e, $(\theta_0, \theta_1, \theta_2, \theta_3) = (\frac{3}{4}, \frac{4}{4}, \frac{3}{4}, \frac{2}{4})$. Note that the indexes of the 4 generators could be permuted arbitrarily and the particular selection of their ordering serves just for better illustration. The other 3 models are given with analogous notation as $(1((23)_{\frac{5}{4}}(4(56)_{\frac{6}{4}})_{\frac{5}{4}})_{\frac{4}{4}})_{\frac{2}{4}}$, $(1((23)_{\frac{5}{4}}(4(5(67)_{\frac{7}{4}})_{\frac{6}{4}})_{\frac{5}{4}})_{\frac{4}{4}})_{\frac{2}{4}}$ and $((1(2(34)_{\frac{5}{4}})_{\frac{4}{4}})_{\frac{3}{4}}((56)_{\frac{4}{4}}(7(89)_{\frac{5}{4}})_{\frac{4}{4}})_{\frac{3}{4}})_{\frac{2}{4}}$. The smallest difference between the parameters is set to $\frac{1}{4}$. As we revealed, while we experimented with different parameterizations, a larger difference in the parameters could hide the impact of the bias of the concerned methods on the structure determination, and the results obtained by different methods can be similar is some of those cases. Setting it to $\frac{1}{4}$ fully reveals the impact of the bias and clearly shows the difference among the methods.

The results for each model are shown in Table 1 and are divided by the double lines. As we are interested in binary copulas, we choose for the comparison the methods θ_{binary}, θ_{RML}, τ_{binary}, which return binary copula structures as their results. The first 2 methods are based on ML estimation technique, whereas the third method is based on the $\theta - \tau$ relationship. To get the results we used their R implementation described in [15]. Our method, implemented in Matlab, is denoted as τ^{avg}_{binary}, i.e., the involved function g is selected to be the avg function. As θ_{RML} failed in most cases for $d \geq 7$, the results for the method for those dimensions are not presented.

Firstly, we assess the ability of the methods to determine the true copula structure correctly. This can be seen from the third and the fourth column. The third column shows 3 the most frequent structures obtained by the method (if the true structure was not the one of the 3 most frequent structures, then we show the 2 most frequent structures and the true structure) with average parameter values. The true structure is emphasized by bold text. The fourth column shows the frequency of the structures. τ^{avg}_{binary} clearly dominates in all four cases ($d = 5, 6, 7, 9$). The other methods show very poor ability to detect

the correct structure, especially for $d \geq 7$, where, e.g., θ_{binary} did not return the correct structure for any among all 100 samples used.

Next, we assess the methods by means of goodness-of-fit. The results can be seen in the fifth and the sixth column, where the statistics $S^{(K)}, S^{(C)}$ (described in [2]) are computed on all bivariate margins and their maximum (the $S^{(K)}, S^{(C)}$ for the worst fitted bivariate margin) is shown. τ^{avg}_{binary} also dominates in all four cases. θ_{RML} shows also good results, but its time consumption for comparable results is considerably higher. The remaining methods show poor results, what is additionally illustrated by the discrepancy between the estimated average parameter values shown in the third column and the true parameter values.

The next two columns show the average Frobenius norm of the difference between the Kendall correlation matrix for the true model and the Kendall correlation matrix for the estimated model and the average Frobenius norm of the difference between the matrix of lower tail coefficients (see [12] for definition) for the true model and the the matrix of lower tail coefficients for the estimated model (as in [13]). The comparison results are similar to the goodness-of-fit comparison. θ_{RML} shows slightly better results than τ^{avg}_{binary} and the remaining methods show significant discrepancy between the theoretical and the empirical quantities.

The last column shows the average computing times needed for a single data sample. τ^{avg}_{binary} is slightly better that the binary methods θ_{binary}, τ_{binary}, whereas θ_{RML} shows significantly higher time consumption, particularly for $d = 6$.

5 Conclusion

Copulas are a feasible tool for the modeling of complex patters. A popular alternative to Gaussian copulas, the hierarchical Archimedean copulas, are convenient copula models even in high dimensions due to their flexibility and rather limited number of parameters. Despite their popularity, a general approach for their estimation is addressed only in one recently published paper, which proposes several methods for the estimation task.

We propose another approach to structure determination and estimation of a hierarchical Archimedean copula, which combines the advantages and avoids the disadvantages of the previously mentioned methods in the terms of the correctly determined structures ratio, the goodness-of-fit of the estimates, and time consumption. This is confirmed in the experiments on simulated data performed for different dimensions and copula models. The proposed method should be preferred to the other mentioned methods and is particularly attractive in applications, where a good approximation and computational efficiency are both crucial issues.

Acknowledgment

The research reported in this paper has been supported by the Czech Science Foundation (GA ČR) grant 13-17187S.

Table 1. The results for the copula models for $d = 5, 6, 7, 9$. The columns contain method names; the 3 most frequent estimated structures with average parameter values; goodness-of-fit statistics $S^{(K)}, S^{(C)}$ (described in [2]); the Frobenius norms of the differences between estimated and true Kendall matrices and lower tail indices; the estimation time in s. The values in parenthesis are the corresponding standard deviations.

d	Method	Structure(s)	%	$S_n^{(K)}$	$S_n^{(C)}$	Avg. error in τ	λ_L	time (in s)
5	θ_{binary}	$(3((12)_{0.77}(45)_{1.01})_{0.76})_{0.24}$	79	2.1478 (0.5043)	0.7206 (0.3205)	0.3101 (0.0253)	0.6306 (0.0416)	0.1517 (0.0382)
		$((\mathbf{12})_{\mathbf{0.69}}(\mathbf{3}(\mathbf{45})_{\mathbf{1.01}})_{\mathbf{0.72}})_{\mathbf{0.68}}$	**18**	0.4897 (0.2050)	0.4089 (0.2107)	0.1426 (0.0245)	0.2893 (0.0496)	
		$((12)_{0.61}(4(35)_{0.85})_{0.71})_{0.61}$	2	0.5546 (0.2206)	0.2843 (0.0421)	0.1208 (0.0180)	0.2346 (0.0369)	
	θ_{RML}	$((\mathbf{12})_{\mathbf{0.71}}(\mathbf{3}(\mathbf{45})_{\mathbf{1.00}})_{\mathbf{0.77}})_{\mathbf{0.54}}$	**52**	0.2102 (0.0826)	0.2426 (0.1078)	0.0511 (0.0222)	0.1016 (0.0473)	0.3616 (0.0560)
		$((45)_{1.01}(3(12)_{0.79})_{0.72})_{0.62}$	43	0.4959 (0.2847)	0.3290 (0.1375)	0.1339 (0.0175)	0.2704 (0.0347)	
		$((12)_{0.80}(4(35)_{0.93})_{0.81})_{0.52}$	3	0.3090 (0.1249)	0.2992 (0.0910)	0.0973 (0.0263)	0.1743 (0.0545)	
	τ_{binary}	$((\mathbf{12})_{\mathbf{0.81}}(\mathbf{3}(\mathbf{45})_{\mathbf{1.04}})_{\mathbf{0.93}})_{\mathbf{0.89}}$	**46**	1.2082 (0.3181)	0.5333 (0.2260)	0.2751 (0.0651)	0.5234 (0.1087)	0.3055 (0.0183)
		$(1(2(3(45)_{1.02})_{0.92})_{0.78})_{0.85}$	23	0.9928 (0.2900)	0.4469 (0.1769)	0.2332 (0.0688)	0.4494 (0.1168)	
		$(2(1(3(45)_{0.99})_{0.92})_{0.79})_{0.88}$	21	0.9659 (0.2024)	0.4022 (0.1625)	0.2443 (0.0457)	0.4709 (0.0799)	
	τ_{binary}^{avg}	$((\mathbf{12})_{\mathbf{0.76}}(\mathbf{3}(\mathbf{45})_{\mathbf{1.01}})_{\mathbf{0.75}})_{\mathbf{0.49}}$	**92**	0.1719 (0.0633)	0.2372 (0.0977)	0.0627 (0.0280)	0.1208 (0.0580)	0.1631 (0.0007)
		$((12)_{0.68}(5(34)_{0.95})_{0.87})_{0.52}$	3	0.1826 (0.0506)	0.2141 (0.0607)	0.0778 (0.0159)	0.1362 (0.0281)	
		$((12)_{0.74}(4(35)_{0.93})_{0.85})_{0.50}$	3	0.2106 (0.0517)	0.3107 (0.1476)	0.0829 (0.0114)	0.1513 (0.0187)	
6	θ_{binary}	$(1(4((23)_{1.28}(56)_{1.53})_{1.28})_{0.55})_{0.18}$	49	2.1014 (0.3962)	0.8661 (0.3472)	0.4078 (0.0338)	0.7367 (0.0545)	0.2674 (0.0784)
		$(\mathbf{1}((\mathbf{23})_{\mathbf{1.16}}(\mathbf{4}(\mathbf{56})_{\mathbf{1.53}})_{\mathbf{1.24}})_{\mathbf{1.15}})_{\mathbf{0.21}}$	**25**	1.1039 (0.2994)	0.4969 (0.2664)	0.2507 (0.0359)	0.4839 (0.0498)	
		$((14)_{0.56}((23)_{1.24}(56)_{1.49})_{1.24})_{0.56}$	22	1.7606 (0.3848)	0.7776 (0.2715)	0.3101 (0.0183)	0.5375 (0.0364)	
	θ_{RML}	$(\mathbf{1}((\mathbf{23})_{\mathbf{1.19}}(\mathbf{4}(\mathbf{56})_{\mathbf{1.53}})_{\mathbf{1.28}})_{\mathbf{1.00}})_{\mathbf{0.50}}$	**48**	0.1965 (0.0681)	0.2945 (0.1197)	0.0506 (0.0187)	0.0884 (0.0401)	3.4299 (2.1311)
		$(1((56)_{1.52}(4(23)_{1.29})_{1.21})_{1.08})_{0.51}$	44	0.3149 (0.1323)	0.3055 (0.1393)	0.1026 (0.0164)	0.1617 (0.0368)	
		$(1(2(3(4(56)_{1.68})_{1.40})_{1.12})_{1.04})_{0.56}$	2	0.2016 (0.0832)	0.3781 (0.0472)	0.1006 (0.0381)	0.1601 (0.0752)	
	τ_{binary}	$(1(2(3(4(56)_{1.56})_{1.49})_{1.39})_{1.39})_{0.70}$	40	0.6187 (0.1551)	0.4378 (0.1623)	0.2478 (0.0605)	0.3970 (0.0995)	0.4983 (0.0205)
		$(1(3(2(4(56)_{1.53})_{1.48})_{1.41})_{1.40})_{0.71}$	32	0.6652 (0.1698)	0.4294 (0.1562)	0.2541 (0.0467)	0.4073 (0.0709)	
		$(\mathbf{1}((\mathbf{23})_{\mathbf{1.37}}(\mathbf{4}(\mathbf{56})_{\mathbf{1.57}})_{\mathbf{1.52}})_{\mathbf{1.36}})_{\mathbf{0.73}}$	**11**	0.6411 (0.1351)	0.4015 (0.1329)	0.2474 (0.0606)	0.4077 (0.0978)	
	τ_{binary}^{avg}	$(\mathbf{1}((\mathbf{23})_{\mathbf{1.27}}(\mathbf{4}(\mathbf{56})_{\mathbf{1.54}})_{\mathbf{1.25}})_{\mathbf{1.00}})_{\mathbf{0.51}}$	**84**	0.1753 (0.0636)	0.2749 (0.1188)	0.0745 (0.0291)	0.1263 (0.0564)	0.2470 (0.0580)
		$(1((23)_{1.21}(5(46)_{1.49})_{1.36})_{1.04})_{0.50}$	4	0.1535 (0.0486)	0.3090 (0.1197)	0.1017 (0.0413)	0.1640 (0.0806)	
		$(1(3(2(4(56)_{1.62})_{1.38})_{1.20})_{1.06})_{0.54}$	3	0.1657 (0.0102)	0.1743 (0.0461)	0.1174 (0.0291)	0.1738 (0.0424)	
7	θ_{binary}	$((14)_{0.52}((23)_{1.24}(5(67)_{1.74})_{1.41})_{1.24})_{0.52}$	48	2.3349 (0.5362)	1.0978 (0.5512)	0.3810 (0.0292)	0.6637 (0.0624)	0.3827 (0.0345)
		$(1(4((23)_{1.25}(5(67)_{1.77})_{1.43})_{1.24})_{0.48})_{0.14}$	18	2.7023 (0.4039)	1.2764 (0.5674)	0.5236 (0.0396)	0.9294 (0.0556)	
		$(1((45)_{1.17}((23)_{1.35}(67)_{1.77})_{1.34})_{1.16})_{0.19}$	16	1.3054 (0.3649)	0.5234 (0.1954)	0.3388 (0.0272)	0.6068 (0.0327)	
	τ_{binary}	$(1(2(3(4(5(67)_{1.79})_{1.73})_{1.63})_{1.46})_{1.45})_{0.70}$	45	0.8215 (0.1864)	0.4797 (0.1671)	0.3173 (0.0740)	0.4776 (0.1128)	0.7435 (0.0217)
		$(1(3(2(4(5(67)_{1.81})_{1.76})_{1.66})_{1.47})_{1.46})_{0.72}$	32	0.8420 (0.2039)	0.5341 (0.1925)	0.3333 (0.0682)	0.5047 (0.1001)	
		$(\mathbf{1}((\mathbf{23})_{\mathbf{1.48}}(\mathbf{4}(\mathbf{5}(\mathbf{67})_{\mathbf{1.85}})_{\mathbf{1.85}})_{\mathbf{1.67}})_{\mathbf{1.48}})_{\mathbf{0.67}}$	**3**	0.8633 (0.1079)	0.4852 (0.1117)	0.3373 (0.1367)	0.5019 (0.1990)	
	τ_{binary}^{avg}	$(\mathbf{1}((\mathbf{23})_{\mathbf{1.27}}(\mathbf{4}(\mathbf{5}(\mathbf{67})_{\mathbf{1.80}})_{\mathbf{1.52}})_{\mathbf{1.25}})_{\mathbf{1.00}})_{\mathbf{0.50}}$	**77**	0.1877 (0.0452)	0.3065 (0.1462)	0.0895 (0.0412)	0.1472 (0.0705)	0.3255 (0.0704)
		$(1((23)_{1.26}(4(7(56)_{1.65})_{1.55})_{1.28})_{1.02})_{0.49}$	6	0.1854 (0.0535)	0.3338 (0.1970)	0.0902 (0.0176)	0.1394 (0.0396)	
		$(1((23)_{1.25}(4(6(57)_{1.55})_{1.42})_{1.25})_{1.02})_{0.50}$	5	0.2094 (0.0800)	0.4709 (0.2645)	0.0951 (0.0265)	0.1514 (0.0599)	
9	θ_{binary}	$((17)_{0.51}((2(34)_{1.25})_{0.90}((56)_{1.02}(89)_{1.26})_{1.02})_{0.89})_{0.51}$	58	1.6487 (0.4330)	0.7410 (0.2550)	0.4771 (0.0440)	0.9144 (0.0823)	0.7862 (0.0565)
		$(1((2(34)_{1.25})_{0.86}((56)_{0.96}(7(89)_{1.33})_{1.01})_{0.96})_{0.86})_{0.13}$	11	3.5263 (0.5750)	0.9699 (0.3482)	0.6364 (0.0340)	1.1800 (0.0544)	
		$(1((56)_{0.91}((2(34)_{1.32})_{0.96}(7(89)_{1.30})_{0.99})_{0.94})_{0.72})_{0.13}$	10	4.1839 (0.4664)	1.2621 (0.4175)	0.6296 (0.0248)	1.1628 (0.0480)	
	τ_{binary}	$((1(2(34)_{1.34})_{1.22})_{1.06}(6(5(7(89)_{1.28})_{1.22})_{1.06})_{1.06})_{1.11}$	15	2.6079 (0.3879)	0.9986 (0.2883)	0.7463 (0.0910)	1.3381 (0.1301)	1.4654 (0.0197)
		$((1(2(34)_{1.31})_{1.24})_{1.12}(5(6(7(89)_{1.29})_{1.23})_{1.12})_{1.11})_{1.12}$	13	2.3948 (0.3476)	0.9770 (0.2834)	0.7620 (0.1053)	1.3583 (0.1537)	
		$((\mathbf{1}(\mathbf{2}(\mathbf{34})_{\mathbf{1.21}})_{\mathbf{1.17}})_{\mathbf{1.04}}((\mathbf{56})_{\mathbf{1.06}}(\mathbf{7}(\mathbf{89})_{\mathbf{1.13}})_{\mathbf{1.10}})_{\mathbf{1.00}})_{\mathbf{1.05}}$	**4**	2.3784 (0.2925)	0.8742 (0.3656)	0.6753 (0.1515)	1.2305 (0.2292)	
	τ_{binary}^{avg}	$((\mathbf{1}(\mathbf{2}(\mathbf{34})_{\mathbf{1.27}})_{\mathbf{0.99}})_{\mathbf{0.75}}((\mathbf{56})_{\mathbf{1.00}}(\mathbf{7}(\mathbf{89})_{\mathbf{1.28}})_{\mathbf{1.01}})_{\mathbf{0.75}})_{\mathbf{0.50}}$	**81**	0.2491 (0.0748)	0.3328 (0.1223)	0.1134 (0.0416)	0.2096 (0.0876)	0.4851 (0.0019)
		$((1(3(24)_{1.17})_{1.07})_{0.72}((56)_{0.97}(7(89)_{1.27})_{0.99})_{0.76})_{0.49}$	4	0.2264 (0.0572)	0.1860 (0.0401)	0.1264 (0.0568)	0.2400 (0.1354)	
		$((1(2(34)_{1.41})_{1.07})_{0.84}((56)_{1.05}(9(78)_{1.26})_{1.12})_{0.83})_{0.56}$	3	0.1921 (0.0297)	0.3401 (0.2103)	0.1444 (0.0219)	0.2576 (0.0446)	

References

1. C. Genest and A. Favre. Everything you always wanted to know about copula modeling but were afraid to ask. *Hydrol. Eng.*, 12:347 – 368, 2007.
2. C. Genest, B. Rémillard, and D. Beaudoin. Goodness-of-fit tests for copulas: A review and a power study. *Insurance: Mathematics and Economics*, 44(2):199 – 213, 2009.
3. Y. González-Fernández and M. Soto. copulaedas: An r package for estimation of distribution algorithms based on copulas. *CoRR*, abs/1209.5429, 2012.
4. J. Górecki and M. Holeňa. An alternative approach to the structure determination of hierarchical archimedean copulas. Mathematical methods in econommics, MME 2013, Jihlava, 2013.
5. M. Hofert. Construction and sampling of nested archimedean copulas. In P. Jaworski, F. Durante, W. K. Hardle, and T. Rychlik, editors, *Copula Theory and Its Applications*, volume 198 of *Lecture Notes in Statistics*, pages 147–160. Springer Berlin Heidelberg, 2010.
6. M. Hofert. *Sampling Nested Archimedean Copulas with Applications to CDO Pricing*. PhD thesis, Ulm University, 2010.
7. M. Hofert. Efficiently sampling nested archimedean copulas. *Computational Statistics and Data Analysis*, 55(1):57 – 70, 2011.
8. S.-C. Kao, A. R. Ganguly, and K. Steinhaeuser. Motivating complex dependence structures in data mining: A case study with anomaly detection in climate. *Data Mining Workshops, International Conference on*, 0:223–230, 2009.
9. I. Kojadinovic. Hierarchical clustering of continuous variables based on the empirical copula process and permutation linkages. *Computational Statistics & Data Analysis*, 54(1):90 – 108, 2010.
10. F. Lascio and S. Giannerini. A copula-based algorithm for discovering patterns of dependent observations. *Journal of Classification*, 29:50–75, 2012.
11. A. J. McNeil and J. Nešlehová. "multivariate archimedean copulas, d-monotone functions and l1-norm symmetric distributions. *The Annals of Statistics*, 37:3059 – 3097, 2009.
12. R. Nelsen. *An Introduction to Copulas*. Springer, 2nd edition, 2006.
13. O. Okhrin, Y. Okhrin, and W. Schmid. On the structure and estimation of hierarchical archimedean copulas. *Journal of Econometrics*, 173(2):189 – 204, 2013.
14. O. Okhrin, Y. Okhrin, and W. Schmid. Properties of hierarchical archimedean copulas. *Statistics & Risk Modeling*, 30(1):21–54, 2013.
15. O. Okhrin and A. Ristig. Hierarchical Archimedean copulae: The HAC package. Discussion paper 2012, 036, CRC 649, Economic Risk, 2012.
16. M. Rey and R. V. Copula mixture model for dependency-seeking clustering. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, Edinburgh, Scotland, UK, 2012.
17. C. Savu and M. Trede. Goodness-of-fit tests for parametric families of archimedean copulas. *Quantitative Finance*, 8(2):109–116, 2008.
18. C. Savu and M. Trede. Hierarchies of archimedean copulas. *Quantitative Finance*, 10:295–304, 2010.
19. A. Sklar. Fonctions de répartition a n dimensions et leurs marges. *Publ. Inst. Stat. Univ. Paris*, 8:229 – 231, 1959.
20. L. Wang, X. Guo, Z. J., and Y. Hong. Copula estimation of distribution algorithms based on exchangeable archimedean copula. *International Journal of Computer Applications in Technology*, 43:13 – 20, 2012.

Developing Personalized Classifiers for Retrieving Music by Mood

Amanda Cohen Mostafavi[1], Zbigniew W. Raś[1,2], and Alicja Wieczorkowska[3]

[1] University of North Carolina, Dept. of Comp. Science, Charlotte NC 28223, USA,
[2] Warsaw University of Technology, Institute of Comp. Science, 00-665 Warsaw, Poland,
[3] Polish-Japanese Institute of Information Technology, 02-008 Warsaw, Poland

Abstract. With the increased amount of music that is available to the average user, either online or through their own collection, there is a need to develop new ways to organize and retrieve music. We propose a system by which we develop a set of personalized emotion classifiers, one for each emotion in a set of 16 and a set unique to each user. We train a set of emotion classifiers using feature data extracted from audio which has been tagged with a set of emotions by volunteers. We then develop SVM, kNN, Random Forest, and C4.5 tree based classifiers for each emotion and determine the ideal classification algorithm. Finally, we compare our personalized emotion classifiers to a set of non-personalized classifiers.

Keywords: Music information retrieval, classification, SVM, kNN, Random Forest, C4.5

1 Introduction

With the average size of a person's digital music collection expanding into the hundreds and thousands, there is a need for creative and efficient ways to search for and index songs. This problem shows up in several sub-areas of music information retrieval such as genre classification, automatic artist identification, and instrument detection. Here we focus on indexing music by emotion, as in how the song makes the listener feel. This way the user could select songs that make him/her happy, sad, excited, depressed, or angry depending on what mood the listener is in (or wishes to be in). However the way a song makes someone feel, or the emotions he associates with the music, varies from person to person for a variety of reasons ranging from personality and taste to upbringing and the music the listener was exposed to growing up. This means that any sort of effective emotion indexing system must be personal and/or adaptive to the user. This is so far a mostly unexplored area of Music Information Retrieval (MIR) research, as many researchers that attempt to personalize their music emotion recognition systems do so from the perspective of finding how likely the song is to be tagged with certain emotions rather than finding a way to create a system that can be personalized.

We present a system through which we can build and train personalized user classifiers, which are unique for individual users. We built these classifiers based on user data accumulated through an online survey and music data collected via a feature extraction toolkit called MIRToolbox [1]. We then use four classification algorithms to determine the ideal algorithm for this data: support vector machines (SVM), k-nearest neighbors (kNN), random forest, and C4.5 trees. We finally take the ideal algorithm and build a broad non-personalized classifier to compare the personalized classifiers to.

2 Related Work

There is some discussion as to the possible usefulness of creating a personalized music recommender system. On the one hand [2] demonstrated that emotion in music is not so subjective that it cannot be modeled; on the other hand the results from researchers who attempt to build personalized music emotion recommendation systems are very promising, suggesting personalization is at least a way to improve emotion classification accuracy. Yang et al in [3] was one of the earliest to study the relationship between music emotion recognition and personality. The authors looked at users demographic information, musical experience, and user scores on the Big Five personality test to determine possible relationships and build their system. Classifiers were built based on support vector regression, and test regressors trained on general data and personalized data. The results were that the personalized regressors outperformed the general regressors in terms of improving accuracy, first spotlighting the problem of trying to create personalized recommendation systems for music and mood based on general groups. However, there has been continued work on collaborative filtering, as well as hybridizing personalized and group based preferences. Lu et al in [4] proposed a system that combined emotion-based, content-based, and collaborative-based recommendation and achieved an overall accuracy of 90%. In [5], the author first proposed the idea of using clustering in order to predict emotions for a group of users. The results were good, but some improvement was needed. The users were clustered into only two groups based on their answers to a set of questions, and the prediction was based on MIDI files rather than real audio. In this work, we propose creating personalized classifiers first (trained on real audio data), clustering users, creating representative classifiers for each cluster, and then allowing the classifiers to be altered based on user behavior.

Each of the possible classification algorithms has been used commonly in previous music information retrieval research, with varying results. kNN had been evaluated previously in [6] and [7] for genre classification. [6] achieved a 90%-98% classification accuracy by combining kNN and Neural Network classifiers and applying them to MIDI files using a 2-level genre hierarchical system. The authors of [7] on the other hand only achieved a 61% accuracy at the highest using real audio and k of 3. Random Forest was used principally in [8] for instrument classification in noisy audio. Sounds were created with one primary instrument and artificial noise of varying levels added in incrementally. The authors found

that the percentage error was overall much lower than previous work done with SVM classifiers on the same sounds up until the noise level in the audio reached 50%. The authors also observed that Random Forest could also indicate the importance of certain attributes in the classification based on the structure of the resulting trees and the attributes used in the splitting. SVM classification is one of the more common algorithms used in music information retrieval for a variety of tasks, such as [9] for mood classification, [10] for artist identification (compared with k-nearest neighbors and Gaussian Mixture Models), and [11] for mood tracking. It has also been evaluated beside other classifiers in [7] for genre identification. These evaluations have shown the SVM classifier to be remarkably accurate, particularly in predicting mood. Regarding C4.5 decision trees, the authors in [12] in a comparison of the J48 implementation of C4.5 to Bayesian network, logical regression, and logically weighted learning classification models for musical instrument classification found that J48 was almost universally the most accurate classifier (regardless of the features used to train the classifier). The classification of musical instrument families (specifically string or woodwind) using J48 ranged in accuracy from 90-92%, and the classification of actual instruments ranged from 60-75% for woodwinds and 60-67% for strings.

3 Data Composition and Collection

3.1 Music Data

Music data was collected from 100 audio clips 25-30 seconds in length culled from one of the author's personal music collection. These clips were split into 12-15 segments (depending on the length of the original clip) of roughly 0.8 seconds in order to allow for changes in annotation as the clip progresses, resulting in a total of 1440 clips. These clips originated from several film and video game sound tracks in order to achieve a similar effect to the dataset composed in [13] (namely a set composed of songs that are less known and more emotionally evocative). As such the music was mainly instrumental with few if any intelligible vocals. The MIRToolbox[1] collection was then used to extract musical features. MIRToolbox is a set of functions developed for use in MATLAB which uses, among others, MATLAB's Signal Processing toolbox. It reads .wav files at a sample rate of 44100 Hz. The following features were extracted using this toolbox.

– **Rhythmic Features** (fluctuation peak, fluctuation centroid, frame-based tempo estimation, autocorrelation, attack time, attack slope): Rhythmic features refer to the set of audio features that describe a song's rhythm and tempo, or how fast the song is, although features such as attack time and attack slope are better indicators of the rhythmic style of the audio rather than pure tempo estimation. Fluctuation based features are based on calculations to a fluctuation summary (calculated from the estimated spectrum with a Bark-band redistribution), while the rest of the rhythmic features are based on the calculation of an onset detection curve (which shows the rhythmic pulses in the song in the form of amplitude peaks for each frame).

– **Timbral Features** (spectral centroid, spectral spread, coefficient of spectral skewness, kurtosis, spectral flux, spectral flatness, irregularity, Mel-Frequency Cepstral Coefficients (MFCC) features, zero crossings, brightness): Timbral features describe a piece's sound quality, or the sonic texture of a piece of audio. The timbre of a song can change based on instrument composition as well as play style. Most of these features are derived from analysis of the audio spectrum, a decomposition of an audio signal. MFCC features are based on analysis of audio frequencies (based on the Mel scale, which replicates how the human ear processes sound). Brightness and zero crossings are calculated based on the audio signal alone.
– **Tonal Features** (pitch, pitch chromogram peak and centroid, key clarity, mode, HCDF): Tonal features describe the tonal aspects of a song such as key, dissonance, and pitch. These features are based primarily off the formulation of a pitch chromogram, which shows the distribution of energy across pitches based on the calculation of dominant frequencies in the audio.

3.2 User Data

We have created a questionnaire so that individuals can go through multiple times and annotate different sets of music based on their moods on a given day. 68 users completed the questionnaire between 1 and 8 times, resulting in almost 400 unique user sessions.

Questionnaire Structure The Questionnaire is split into 5 sections

– Demographic Information (where the user is from, age, gender, ethnicity)
– General Interests (favorite books, movies, hobbies)
– Musical Tastes (what music the user generally likes, what he listens to in various moods)
– Mood Information (a list of questions based on the Profile of Mood States)
– Music Annotation (where the user annotates a selection of musical pieces based on mood)

The demographic information section is meant to compose a general picture of the user. The questions included ask for ethnicity (based on the NSF definitions), age, what level of education the user has achieved, what field they work or study in, where the user was born, and where the user currently lives. Also included is whether the user has ever lived in a country other than where he/she was born or where he/she currently lives for more than three years. This question is included because living in another country for that long would expose the user to music from that country (see figure 1).

The general interests section gathers information on the user's interests outside of music. It asks for the user's favorite genre of books, movies, and what kind of hobbies he/she enjoys. It also asks whether the user enjoyed math in school, whether he/she has a pet or would want one, whether he/she believes in

an afterlife, and how he/she would handle an aged parent. These questions are all meant to build a more general picture of the user (see figure 2).

The musical tastes section is meant to get a better picture of how the user relates to music. It asks how many years of formal musical training the user has had, as well as his/her level of proficiency in reading or playing music if any. It also asks what genre of music the user listens to when they're happy, sad, angry, and calm (see figure 3).

The mood information section is a shortened version of the Profile of Mood States [14]. The Profile of Mood States asks users to rate how strongly he/she has been feeling a set of emotions over a period of time from the following list of possible responses:

- Not at all
- A little
- Moderately
- Quite a bit
- Extremely

The possible emotions asked about in the mood information session are listed below:

- Tense
- Shaky
- Uneasy
- Sad
- Unworthy
- Discouraged
- Angry
- Grouchy
- Annoyed
- Lively
- Active
- Energetic
- Worn Out
- Fatigued
- Exhausted
- Confused
- Muddled
- Efficient

A sample of these questions can be seen in figure 4. This is the section that is filled out every time the user returns to annotate music, since their mood would affect how they annotate music on a given day. These answers are later converted into a mood vector for each session, which describes the user's mood state at the time of the session.

Finally, the music annotation section is where users go to annotate a selection of clips. 40 clips are selected randomly from the set of 1440 clips mentioned in

section 3.1. The user is then asked to check the checkbox for the emotion he/she feels in the music, along with a rating from 1-3 signifying how strongly the user feels that emotion (1 being very little, 3 being very strongly). The user has a choice of 16 possible emotions to pick (to be specific, 12 emotions and 4 generalizations), based on a 2-D hierarchical emotional plane (see figure 6 for the emotion plane and figure 5 for a view of the questionnaire annotation section).

When the user goes through the questionnaire any time after the first time, he only has to fill out the mood profile and the annotations again. Each of these separate sections (along with the rest of the corresponding information) is treated as a separate user, so each individual session has classifiers trained for each emotion, resulting in 16 emotion classifiers for each user session.

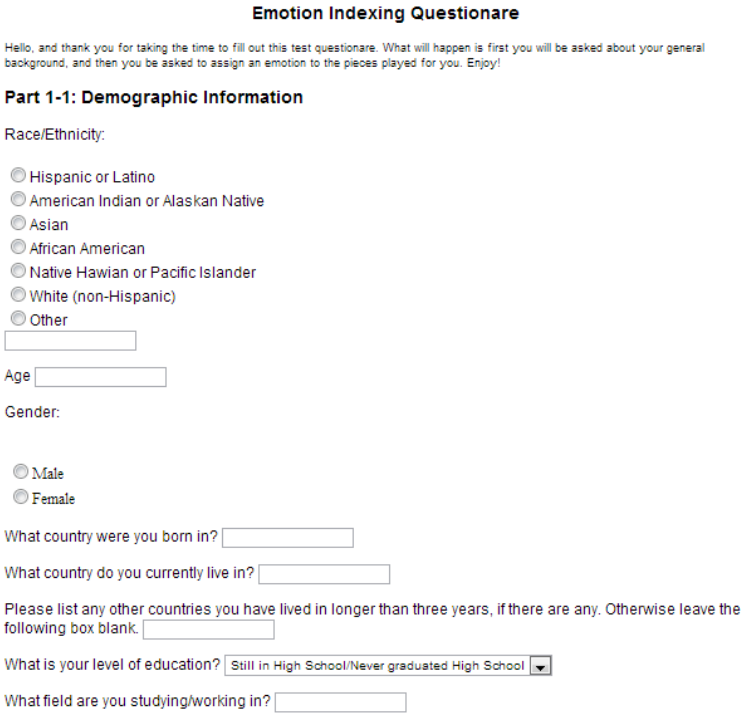


Fig. 1. The demographic information section of the questionnaire

Emotion Model This model was first presented in [5], and implements a hierarchy on the 2-dimensional emotion model, while also implementing discrete elements. The 12 possible emotions are derived from various areas of the 2-dimensional arousal-valence plane (based on Thayer's 2 dimensional model of arousal and valence [15]). However there are also generalizations for each area of

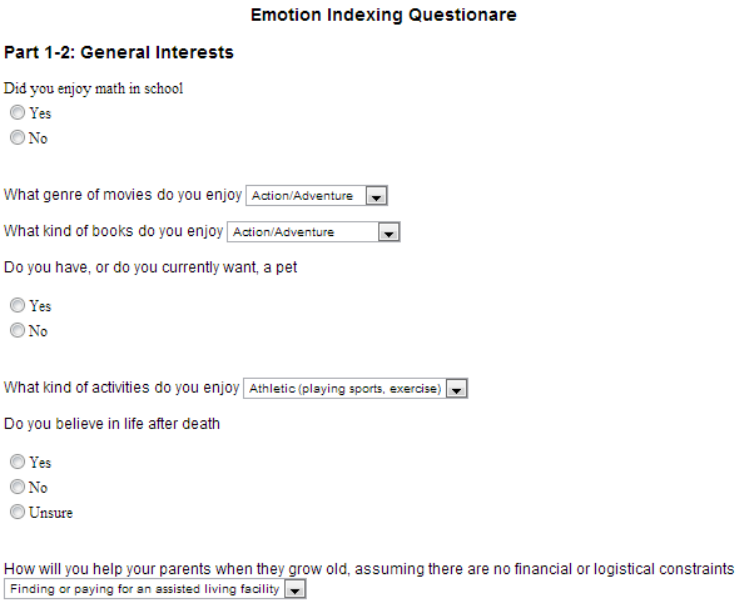


Fig. 2. The general interests section of the questionnaire

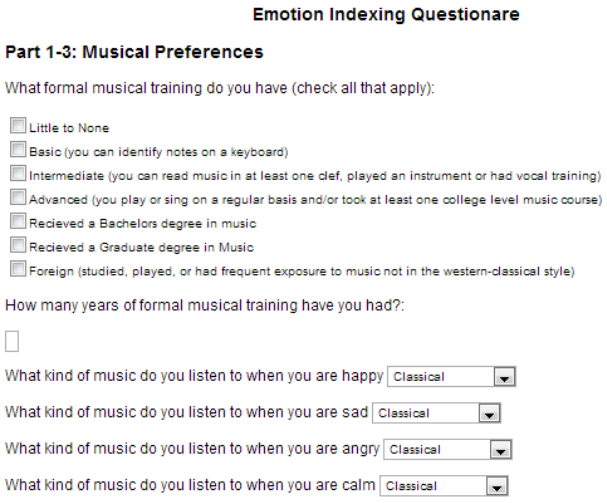
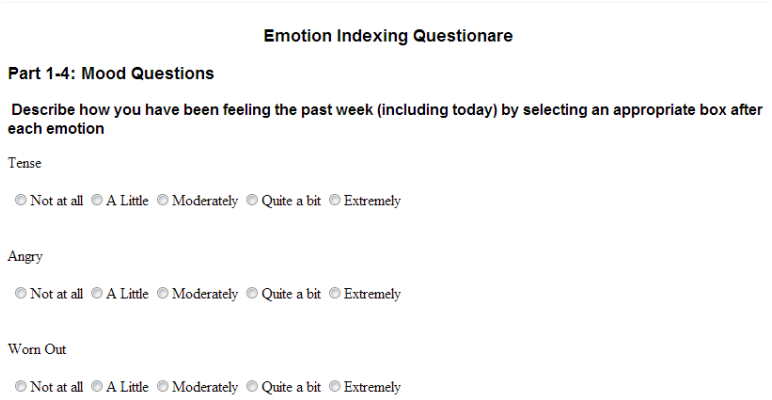


Fig. 3. The musical taste/background information section of the questionnaire



Emotion Indexing Questionare

Part 1-4: Mood Questions

Describe how you have been feeling the past week (including today) by selecting an appropriate box after each emotion

Tense

Not at all | A Little | Moderately | Quite a bit | Extremely

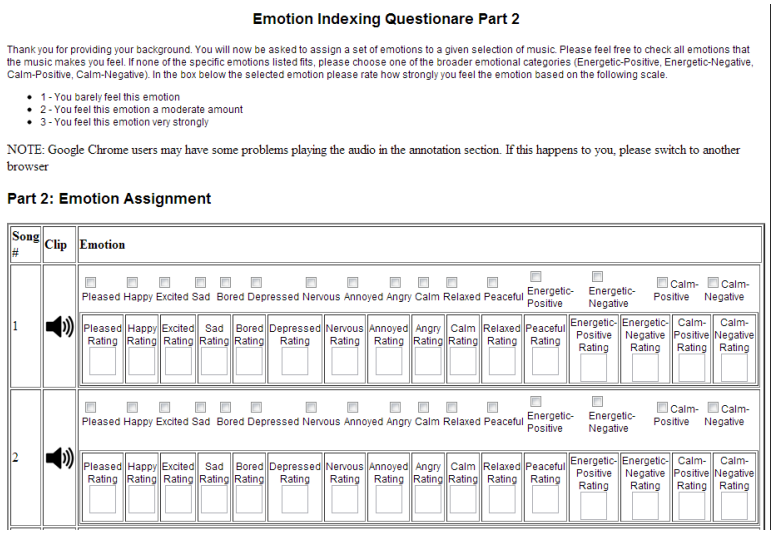
Angry

Not at all | A Little | Moderately | Quite a bit | Extremely

Worn Out

Not at all | A Little | Moderately | Quite a bit | Extremely

Fig. 4. Part of the mood state information section of the questionnaire. This section is filled out every time the user reenters the questionnaire (the user starts on this page once he/she has filled out the rest of the questionnaire once)



Emotion Indexing Questionare Part 2

Thank you for providing your background. You will now be asked to assign a set of emotions to a given selection of music. Please feel free to check all emotions that the music makes you feel. If none of the specific emotions listed fits, please choose one of the broader emotional categories (Energetic-Positive, Energetic-Negative, Calm-Positive, Calm-Negative). In the box below the selected emotion please rate how strongly you feel the emotion based on the following scale.

- 1 - You barely feel this emotion
- 2 - You feel this emotion a moderate amount
- 3 - You feel this emotion very strongly

NOTE: Google Chrome users may have some problems playing the audio in the annotation section. If this happens to you, please switch to another browser

Part 2: Emotion Assignment

Song #	Clip	Emotion
1		Pleased, Happy, Excited, Sad, Bored, Depressed, Nervous, Annoyed, Angry, Calm, Relaxed, Peaceful, Energetic-Positive, Energetic-Negative, Calm-Positive, Calm-Negative
		Pleased Rating, Happy Rating, Excited Rating, Sad Rating, Bored Rating, Depressed Rating, Nervous Rating, Annoyed Rating, Angry Rating, Calm Rating, Relaxed Rating, Peaceful Rating, Energetic-Positive Rating, Energetic-Negative Rating, Calm-Positive Rating, Calm-Negative Rating
2		Pleased, Happy, Excited, Sad, Bored, Depressed, Nervous, Annoyed, Angry, Calm, Relaxed, Peaceful, Energetic-Positive, Energetic-Negative, Calm-Positive, Calm-Negative
		Pleased Rating, Happy Rating, Excited Rating, Sad Rating, Bored Rating, Depressed Rating, Nervous Rating, Annoyed Rating, Angry Rating, Calm Rating, Relaxed Rating, Peaceful Rating, Energetic-Positive Rating, Energetic-Negative Rating, Calm-Positive Rating, Calm-Negative Rating

Fig. 5. The music emotion annotation section, also filled out every time the user goes through the questionnaire. The user clicks on a speaker to hear a music clip, then checks an emotion and supplies a rating 1 to 3

the plane (excited-positive, excited-negative, calm-positive, and calm-negative) that the users can select as well. This compensates for songs that might be more ambiguous to the user; if a user generally knows that a song is high-energy and positive feeling but the words excited, happy, or pleased don't adequately describe it, they can select the generalization of energetic-positive.

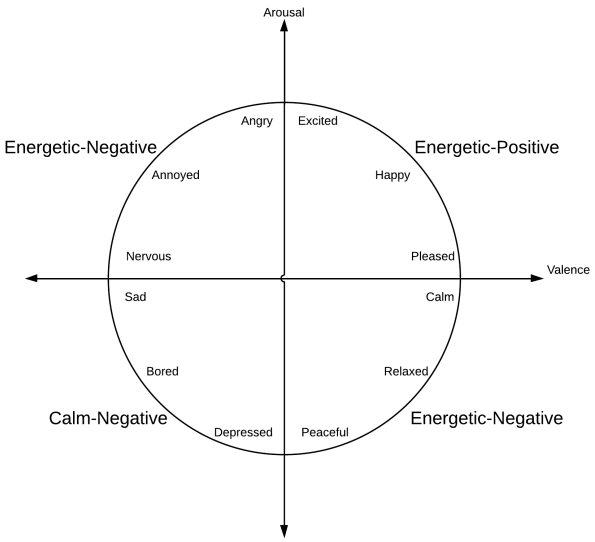


Fig. 6. A diagram of the emotional model to be used for classifier clustering

3.3 Classifier Development

Personalized classifiers were trained and tested using the classification algorithms listed previously (C4.5, SVM, Random Forest, kNN). The user annotation data was first converted so that each annotation for each song was represented as a vector of 16 numbers with each number representing the emotion labeling. The numbers ranged from 0 to 3, with 0 representing an emotion that was not selected by the user and the remaining numbers being the strength the user entered with the annotation. These vectors for all the users were then linked with the feature data extracted from the corresponding music clips. From this resulting table all the annotations and music data linked with individual user IDs were separated and used to train and test personalized classifiers for each emotion. This resulted in each user having at most 16 personalized classifiers (depending on whether the user used a given emotion during the course of annotating), where for each classifier the class attribute was one of the 16 possible emotions.

The classifiers were all evaluated via Weka[16] using 10-fold cross validation. For the C4.5 classifier we used the J48 implementation in Weka and for kNN we used Weka's IBk implementation. Analysis of these results indicates which classifier algorithm is most effective for personalized classification and, therefore, the most effective cluster-driven classifier.

4 Results

All four classifiers achieved a relatively high average accuracy, all above 80%. SVM achieved the lowest accuracy, 82.35%, while J48 trees achieved the highest accuracy, 86.62%. However, SVM as well as Random Forest achieved the highest average F-score (a combined measure of precision and recall), implying a higher precision and recall for those classifiers. IBk on the other hand had the lowest F-score of 0.92. SVM was expected to have a higher accuracy since it works so well with music data, but our previous success with J48 means the high accuracies and F-scores are not surprising.

Looking at the average Kappa statistic reveals further insights into the effectiveness of each classifier. The Kappa statistic measures the agreement between a true class and the prediction, and the closer to 1 the statistic is the more agreement (1 represents complete agreement). None of the classifiers reaches higher than 0.1, although again IBk has the highest average Kappa (J48, again, the lowest). This all suggests that while J48 is overall very accurate it is more inconsistent in terms of this particular set of data, while SVM is moderately accurate and very consistent.

Table 1. Table of classifier accuracies and F-scores

Classifier	Average Accuracy	Average F-Score	Average Kappa
SVM	82.35%	0.90	0.137658
IBk	85.7%	0.87	0.153468
J48	86.62%	0.89	0.076869
Random Forest	84.25%	0.90	0.133027

4.1 Comparison with Non-Personalized Classifiers

As it proved to be the most accurate classifier, we have chosen J48 as the algorithm to use to build the non-personalized classifiers for comparison. We again built 16 emotion classifiers, this time using all the user annotations to train and test rather than individual user annotations. The results compared to the personalized J48 classifiers are shown in Table 2.

The average accuracy doesn't change too much between personalized and non-personalized classifiers (only 0.7%), however this was mainly due to the fact that several of the emotions were not used to the same extent as others

Table 2. Comparison of classifier accuracies and F-scores between personalized and non-personalized classifiers

Classifier	Average Accuracy	Average F-Score	Average Kappa
Personalized J48	86.62%	0.89	0.076869
Non-personalized J48	87.29%	0.82	0.00015

when tagging (for example, the generalized emotions), and in that case all the classifier did was predict '0' (for emotions that were not selected). This raised the accuracy for those classifiers, but it's not nearly as indicative as to the quality of the classifier as the F-Score and Kappa, which showed a great deal of improvement in the personalized classifier. The average F-score for the non-personalized classifiers is 0.08 less than the average F-score for personalized classifiers, and the average Kappa for the non-personalized classifiers is far less than the personalized classifiers. These both signify a significant loss in classifier consistency once the classifiers are no longer personalized.

5 Conclusion

We have presented a system through which we build personalized music emotion classifiers based on user data accumulated through an online survey. Using this data, we have been able to build classifiers that are about as accurate as standard, non-personalized classifiers but far more consistent. In a real world situation, this would mean that music that accurately reflects the user's mood (or desired mood) would be recommended far more often than not. Future work would involve using these classifiers in a full music player, and improving classifiers even further by clustering users.

References

1. Lartillot, O., Toiviainen, P., Eerola, T.: MIRtoolbox. University of Jyväskylä (2008)
2. Laurier, C., Herrera, P.: Automatic Detection of Emotion in Music: Interaction with Emotionally Sensitive Machines. In Vallverdu, D., Casacuberta, D., eds.: Handbook of Research on Synthetic Emotions and Sociable Robotics: New Applications in Affective Computing and Artificial Intelligence. IGI Global (2009) 9–32
3. Yang, Y.H., Su, Y.F., Lin, Y.C., Chen, H.H.: Music Emotion Recognition: The Role of Individuality. In: Proc. International Workshop on Human-centered Multimedia 2007 {(HCM'07)}, Bavaria, Germany, ACM (September 2007)
4. Lu, C.C., Tseng, V.S.: A novel method for personalized music recommendation. Expert Systems with Applications **36**(6) (August 2009) 10035–10044
5. Grekow, J., Ras, Z.W.: Detecting Emotions in Classical Music from MIDI Files. In Rauch, J., Ras, Z., Berka, P., Elomaa, T., eds.: Foundations of Intelligent Systems, Proceedings of ISMIS'09. Volume 5722 of LNAI., Springer (2009) 261–270

6. Mckay, C., Fujinaga, I.: Automatic genre classification using large high-level musical feature sets. IN INT. CONF. ON MUSIC INFORMATION RETRIEVAL, ISMIR 2004 (2004) 525 – 530
7. Silla Jr., C.N., Koerich, A.L., Kaestner, C.A.A.: A Machine Learning Approach to Automatic Music Genre Classification. Journal of the Brazilian Computer Society **14**(3) (2008) 7–18
8. Kursa, M., Rudnicki, W., Wieczorkowska, A., Kubera, E., Kubik-Komar, A.: Musical Instruments in Random Forest. Foundations of Intelligent Systems **5722** (2009) 281–290
9. Laurier, C., Meyers, O., Marxer, R., Bogdanov, D., Serrà, J., Gómez, E., Herrera, P., Wack, N.: Music classification using high-level models. ISMIR 2009 (2010)
10. Mandel, M.I., Ellis, D.P.W.: Song-level features and support vector machines for music classification. In Reiss, J.D., Wiggins, G.A., eds.: Proceedings of the 6th International Conference on Music Information Retrieval - ISMIR 2005. Volume 6., London, U.K. (2005) 594–599
11. Panda, R., Paiva, R.P.: Using Support Vector Machines for Automatic Mood Tracking in Audio Music. In: 130th Audio Engineering Society Convention. (2011)
12. Zhang, X., Ras, Z.: Differentiated harmonic feature analysis on music information retrieval for instrument recognition. (2006) 578–581
13. Eerola, T., Vuoskoski, J.K.: A comparison of the discrete and dimensional models of emotion in music. Psychology of Music **39**(1) (August 2011) 18–49
14. McNair, D.M., Lorr, M., Droppleman, L.F.: Profile of Mood States (POMS) (1971)
15. Thayer, R.E.: The biopsychology of mood and arousal. Oxford University Press (1989)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software. ACM SIGKDD Explorations Newsletter **11**(1) (November 2009) 10

AGWAN: A Generative Model for Labelled, Weighted Graphs

Michael Davis, Weiru Liu, and Paul Miller

Centre for Secure Information Technologies (CSIT),
School of Electronics, Electrical Engineering and Computer Science,
Queen's University, Belfast, United Kingdom
{mdavis05, w.liu}@qub.ac.uk, p.miller@ecit.qub.ac.uk

Abstract. Real-world graphs or networks tend to exhibit a well-known set of properties, such as heavy-tailed degree distributions, clustering and community formation. Much effort has been directed into creating realistic and tractable models for unlabelled graphs, which has yielded insights into graph structure and evolution. Recently, attention has moved to creating models for labelled graphs: many real-world graphs are labelled with both discrete and numeric attributes. In this paper, we present AGWAN (Attribute Graphs: Weighted and Numeric), a generative model for random graphs with discrete labels and weighted edges. The model is easily generalised to edges labelled with an arbitrary number of numeric attributes. We include algorithms for fitting the parameters of the AGWAN model to real-world graphs and for generating random graphs from the model. Using the Enron "who communicates with whom" social graph, we compare our approach to state-of-the-art random labelled graph generators and draw conclusions about the contribution of discrete vertex labels and edge weights to the structure of real-world graphs.

Keywords: Network models, graph generators, random graphs, labelled graphs, weighted graphs, graph mining

1 Introduction

Network analysis is concerned with finding patterns and anomalies in real-world graphs, such as social networks, computer and communication networks, or biological and ecological processes. Real graphs exhibit a number of interesting structural and evolutionary properties, such as power-law or log-normal degree distribution, small diameter, shrinking diameter, and the Densification Power Law [4, 13, 15].

Besides discovering network properties, researchers are interested in the mechanisms of network formation. Generative graph models provide an abstraction of how graphs form; if the model is accurate, generated graphs will obey the same properties as real graphs. Generated graphs are also useful for simulation experiments, hypothesis testing and making predictions about graph evolution or missing graph elements. Most existing models are for unlabelled, unweighted graphs [4, 13]; a generative model for random graphs with discrete labels has recently been proposed [11].

In this paper, we present AGWAN, a generative model for labelled, weighted graphs. Weights are commonly used to represent the number of occurrences of each edge: the

number of e-mails sent between individuals in a social network [1]; the number of calls to a subroutine in a software call graph [6]; or the number of people walking between a pair of door sensors in a building access control network [5]. In other applications, the edge weight may represent continuous values: donation amounts in a bipartite graph of donors and political candidates [1]; distance or speed in a transportation network [6]; or elapsed time between the sensors in the building network [5]. In some cases, the weight is a multi-dimensional feature vector [5, 6].

Our main motivation for this work is to create a model to better understand the laws governing the relationship between graph structure and numeric labels or weights. Furthermore, we want to be able to create realistic random, labelled, weighted graphs for large-scale simulation experiments for our pattern discovery algorithms [5]. Our experiments in §5 show the extent to which various graph properties are related to labels and weights, and measure exactly how "realistic" our random graphs are. Graphs generated with AGWAN are shown to have more realistic degree strength distributions and spectral properties than the comparative methods.

This paper is arranged as follows: §2 is an overview of generative graph models; §3 presents AGWAN, our generative model for weighted and numeric labelled graphs. We include a fitting algorithm to learn AGWAN's parameters from a real input graph, and an algorithm to generate random graphs from the model. §4 gives an overview of the Enron "who communicates with whom" social graph that we use in the experiments, and outlines the statistical measures and tests that we use to evaluate the generated graphs. The experiments in §5 demonstrate that the vertex labels and edge weights of a graph can predict the graph structure with high accuracy. Conclusions are in §6.

2 Related Work

Our understanding of the mathematical properties of graph structure was pioneered by Paul Erdős and Alfréd Rényi [7]. Graph formation is modelled as a Bernoulli process, parameterised by the number of vertices and a wiring probability between each vertex pair. While it has been essential to our understanding of component sizes and expected diameter, the Erdős-Rényi model does not explain other important properties of real-world graphs such as degree distribution, transitivity and clustering [4, 15].

Barabási and Albert's Preferential Attachment model [2] uses the "rich get richer" principle to grow graphs from a few vertices up to the desired size. The probability of an edge is proportional to the number of edges already connected to a vertex. This generates graphs with power-law degree distributions. A number of variants of Preferential Attachment have been proposed [4, 15]; notably, the Forest Fire model [14] which exhibits the Densification Power Law and shrinking diameter effect. Still, Preferential Attachment models lack some desired properties, such as community structure.

The RMat algorithm [4] solves the community structure problem with its recursive matrix approach. RMat graphs consist of 2^n vertices and E edges, with four probabilities a, b, c, d to determine in which quadrant of the adjacency matrix each edge falls. These parameters allow the specification of power-law or log-normal degree distributions; if $a = b = c = d$, the result will be an Erdős-Rényi graph. However, RMat does not model the Densification Power Law and shrinking diameter effect.

Kronecker Graphs [13] are a generalisation of RMat graphs which fulfil all the properties mentioned above. The model starts with an initiator matrix. Kronecker (Tensor) multipication is recursively applied to yield the final adjacency matrix of the desired size. This work synthesises the previous work in random graphs in a very elegant way and proves that RMat graphs are a special case of Stochastic Kronecker graphs. While they fulfil the desired structural properties, Kronecker Graphs are unlabelled.

The Multiplicative Attribute Graph (MAG) model [11] is a generative model for labelled graphs. MAG is parameterised by the number of vertices, a set of prior probabilities for vertex label values and a set of *affinity matrices* specifying the probability of an edge conditioned on the vertex labels. The affinity matrices can be learned from real graphs using Maximum Likelihood Estimation [10]. [11] proves that Kronecker Graphs are a special case of MAG graphs, and that suitably-parameterized MAG graphs fulfil all the desired properties: log-normal or power-law degree distribution, small diameter, the existence of a unique giant component and the Densification Power Law. The MAG model considers discrete vertex labels only. We believe that our method, described in the next section, is the first generative model to include numeric labels or weights.

3 AGWAN: A Generative Model for Labelled, Weighted Graphs

In this section, we present our generative model, AGWAN (Attribute Graph: Weighted and Numeric). The model is illustrated in Fig. 1 for the Enron graph described in §4.

Consider a graph $G = (V, E)$ with discrete vertex label values drawn from a set L. In Fig. 1, $u, v \in V$ are vertices and $w_{uv}, w_{vu} \in \mathbb{R}$ are edge weights. Edges $e \in E$ are specified as a 3-tuple $\langle u, v, w_{uv} \rangle$. In the discussion which follows, we restrict ourselves to a single label on each vertex; we outline how this can be extended to multiple labels in §3.3.

We assume that the edge weights $W^{ij} = \{w_{ij}\}$ follow an arbitrary probability distribution. W^{ij} may be Gaussian, if it is drawn from an independent and identically distributed random variable such as speed, elapsed time, or amounts of money. If W^{ij} represents the number of events occurring in a fixed interval of time—number of e-mails sent or number of people moving between sensors—it is expected to follow a Poisson distribution. Other quantities—frequency of word use or number of papers written by scientists—follow a power-law distribution [15]. In many cases, W^{ij} arises from multiple processes. As the underlying distribution is unknown, we model W^{ij} using a Gaussian Mixture Model (GMM) with an arbitrary number of mixtures. This provides a reasonable approximation to any general probability distribution. We avoid the problem of knowing the "correct" number of mixtures by assuming that W^{ij} consists of an infinite number of mixtures and using variational inference to determine the optimal number for our model [3].

The AGWAN model is parameterised by μ, a set of prior probabilities over L; and Θ, a set of edge weight mixture parameters: $\Theta = \{\Omega^{ij} | i, j \in L\}$. For directed graphs, $|\Theta| = |L|^2$ and we need to generate both w_{uv} and w_{vu} (see Fig. 1). For undirected graphs, $\Omega^{ij} = \Omega^{ji}$, so $|\Theta| = O(|L|^2/2)$ and $w_{vu} = w_{uv}$.

For each combination of vertex attributes $\langle i, j \rangle$, the corresponding mixture model Ω^{ij} parameterises the distribution of edge weights (with an edge weight of 0 indicating no edge). Ω^{ij} is a GMM with M mixtures:

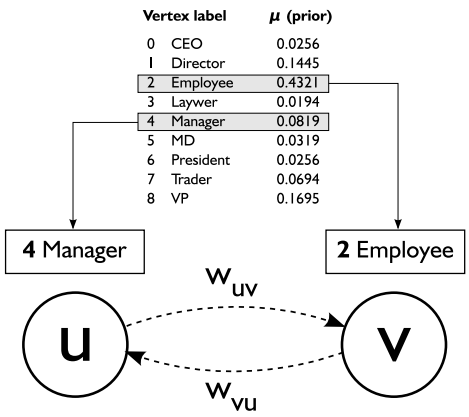


Fig. 1: AGWAN parameters. Vertex labels are selected according to prior probability μ. Edge weight w_{uv} is selected from mixture model Ω^{42} and w_{vu} is selected from mixture model Ω^{24}.

$$\Omega^{ij} = \sum_{m=0}^{M-1} \omega_m^{ij} \cdot \eta(\mu_m^{ij}, (\sigma^2)_m^{ij}) \tag{1}$$

where ω_m^{ij} is the weight of each mixture and $\eta(\mu_m^{ij}, (\sigma^2)_m^{ij})$ is the Gaussian PDF with mean μ_m^{ij} and variance $(\sigma^2)_m^{ij}$. The mixture weights form a probability distribution over the mixtures: $\sum_{m=0}^{M-1} \omega_m^{ij} = 1$. We can specify Ω^{ij} such that the first mixture encodes the probability of no edge: $\omega_0^{ij} = 1 - P(e_{ij})$, where $P(e_{ij})$ is the probability of an edge between pairs of vertices with labels $\langle i, j \rangle$. The model degenerates to an unweighted graph if there are two mixtures, $\eta_0(0,0)$ and $\eta_1(1,0)$. Furthermore, if the weights ω_m^{ij} are the same for all $\langle i, j \rangle$, the model degenerates to an Erdős-Rényi random graph.

3.1 Graph Generation

Algorithm 1 describes how to generate a random graph using AGWAN(N, L, μ, Θ). The number of vertices in the generated graph is specified by N. After assigning discrete label values to each vertex (lines 2–3, *cf.* Fig. 1), the algorithm checks each vertex pair $\langle u, v \rangle$ for the occurrence of an edge (lines 4–7). If $m = 0$, $\langle u, v \rangle$ is not an edge (line 7). If there is an edge, we assign its weight from mixture m (lines 8–9). The generated graph is returned as $G = (V, E)$.

3.2 Parameter Fitting

To create realistic random graphs, we need to learn the parameters μ, Θ from a real-world input graph G. Let W^{ij} be the set of edge weights between pairs of vertices with labels $\langle i, j \rangle$. During parameter fitting, we want to create a model Ω^{ij} for each W^{ij} in G. Each GMM Ω^{ij} has a finite number of mixtures M. If M is known, Ω^{ij} can be estimated using Expectation Maximisation [9]. However, not only is M unknown, but we expect that it will be different for each Ω^{ij} within a given graph model [5].

We solve this problem by modelling Ω^{ij} as a non-parametric mixture model with an unbounded number of mixtures: a Dirichlet Process Gaussian Mixture Model [3]

Algorithm 1 AGWAN Graph Generation

Require: N (no. of vertices), L (set of discrete label values), μ (prior distribution over L), $\Theta = \left\{\Omega^{ij}\right\}$ (set of mixture models)

1: Create vertex set V of cardinality N, edge set $E = \emptyset$
2: **for all** $u \in V$ **do**
3: Assign discrete label $l_u \in L$ from prior μ
4: **for all** $u, v \in V : u \neq v$ **do**
5: $i = l_u, j = l_v$
6: Select Gaussian m uniformly at random from Ω^{ij}
7: **if** $m \neq 0$ **then**
8: Assign edge weight w_{uv} uniformly at random from $\eta(\mu_m^{ij}, (\sigma^2)_m^{ij})$
9: Create edge $e = \langle u, v, w_{uv} \rangle, E = E \cup \{e\}$
return $G = (V, E)$

(DPGMM). "Non-parametric" does not mean that the model has no parameters; rather, the number of parameters is allowed to grow as more data are observed. In essence, the DPGMM is a probability distribution over the probability distributions of the model.

The Dirichlet Process (DP) over edge weights W^{ij} is a stochastic process $DP(\alpha, H_0)$, where α is a positive scaling parameter and H_0 is a finite measure on W^{ij}. If we draw a sample from $DP(\alpha, H_0)$, the result is a random distribution over values drawn from H_0. This distribution H is discrete, represented as an infinite sum of atomic measures. If H_0 is continuous, then the infinite set of probabilities corresponding to the frequency of each possible value that H can return are distributed according to a *stick-breaking process*. The stick-breaking representation of H is given as:

$$\omega_m^{ij}(\mathbf{x}) \prod_{n=1}^{m-1} (1 - \omega_n^{ij}) \qquad\qquad H = \sum_{n=1}^{\infty} \omega_n^{ij}(\mathbf{x}) \delta_{\eta_m^*} \tag{2}$$

where $\{\eta_1^*, \eta_2^*, \ldots\}$ are the atoms representing the mixture components. We learn the mixture parameters using the variational inference algorithm for generating Dirichlet Process Mixtures described in [3]. The weights of each component are generated one-at-a-time by the stick-breaking process, which tends to return the components with the largest weights first. In our experiments, the first 3–5 mixtures accounted for over 99% of the data. Mixtures with weights summing to less than 0.01 are dropped from the model, and the remaining weights $\{\omega_m^{ij}\}$ are normalised.

Algorithm 2 is the algorithm for AGWAN parameter fitting. First, we estimate the vertex priors (lines 1–3). Next, we sample the edge weights for each possible combination of vertex label values, with no edge counting as a weight of zero (lines 4–7). Finally, we estimate the GMMs Ω^{ij} from the appropriate set of samples W^{ij} using the the stick-breaking process described above.

3.3 Extending AGWAN to multiple attributes

We have presented AGWAN for a single discrete vertex label and a single numeric edge label (the weight). Many graphs have multiple labels on vertices and edges. AGWAN can be extended to multiple numeric edge labels by generalising the concept of edge weight to k dimensions. In this case, the mean of each mixture becomes a k-dimensional vector

Algorithm 2 AGWAN Parameter Fitting

Require: Input graph $G = (V, E)$

$L = \{\text{discrete vertex label values}\}$, $d = |L|$

Calculate vertex label priors, apply Laplace smoothing $\forall l \in L : P(l) = \frac{count(l)+\alpha}{N+\alpha d}$

$\mu =$ the normalised probability distribution over L such that $\sum_{i=1}^{d} P(l_i) = 1$

$\forall i, j \in L : W^{ij} = \emptyset$

for all $u, v \in V : u \neq v$ **do**

    $i = l_u, j = l_v$

    $W^{ij} = W^{ij} \cup \{w_{uv}\}$ ▷ If $\langle u, v \rangle$ is not an edge, then w_{uv} has value zero

for all $i, j \in L$ **do**

    estimate Ω^{ij} from W^{ij} using variational inference

$\Theta = \left\{\Omega^{ij}\right\}$

return μ, Θ

and the variance $(\sigma_m^{ij})^2$ is replaced with the $k \times k$ covariance matrix Σ_m^{ij}. The variational algorithm can be accelerated for higher-dimensional data using a kd-tree [12] and has been demonstrated to work efficiently on datasets of hundreds of dimensions.

A more difficult question is how to extend the model to multiple discrete vertex labels. With even a small number of labels, modelling the full joint probability across all possible combinations of label values becomes a complex combinatorial problem with hundreds or thousands of parameters. The MAG model reduces this complexity by assuming that vertex labels are independent, so edge probabilities can be computed as the product of the probabilities from each label [11]. For latent attributes, MAGFIT enforces independence by regularising the variational parameters using mutual information [10]. However, the MAG model has not solved this problem for real attributes, where independence cannot be assumed. Furthermore, multiplying the probabilities sets an upper limit (proportional to $\log N$) on the number of attributes which can be used in the model. In our experiments (§5), using three latent attributes created a less accurate model than using two.

An alternative to multiplying independent probabilities is to calculate the GMM for each edge as the weighted summation of the GMM for each individual attribute; it is likely that some attributes have a large influence on graph structure while others affect it little or not at all. This problem remains a topic for further research; see §6.

4 Experiments

We evaluate our approach by comparing AGWAN with the state-of-the-art in labelled graph generation, represented by the MAG model [10, 11]. We learn the AGWAN and MAG model parameters from a real-world graph. We then generate random graphs from each model and calculate a series of statistics on each graph. These statistics are used to compare how closely the model maps to the input graph.

For our input graph, we use the "who communicates with whom" graph of the Enron e-mail corpus [1] (Fig. 2). The graph has 159 vertices representing Enron's senior employees and 2667 edges representing e-mail communications sent between them. Vertices have a single discrete label which describes the job role of the employee.

The edges are weighted according to the number of e-mails sent between each pair of employees. As e-mail communications are not symmetric, the edges are directed.

We evaluated AGWAN against the following models:

Erdős-Rényi random graph (ER): The ER model $G(n,p)$ has two parameters. We set the number of vertices $n = 159$ and the edge probability $p = 0.106$ to match the input graph as closely as possible. We do not expect a very close fit, but the ER model provides a useful baseline.

MAG with real attributes (MAG-R1): The MAG model with one real attribute is similar to AGWAN with one real attribute, with the difference that $\Theta = \{\Omega^{ij}\}$ is replaced with a set of binary edge probabilities, $\Theta = \{p^{ij}\}$.

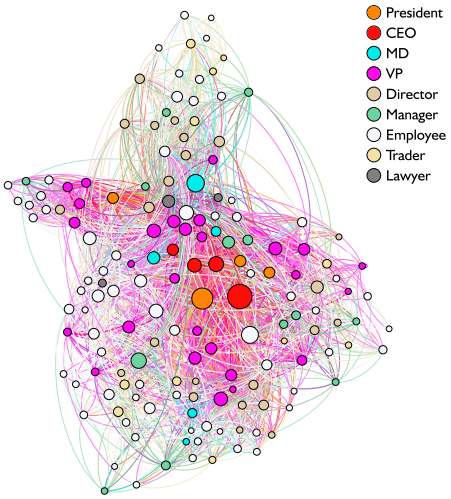


Fig. 2: Social graph of who communicates with whom in the Enron corpus

MAG with latent attributes (MAG-L1, MAG-L2, MAG-L3): The MAG model allows for modelling the graph structure using latent attributes. These models ignore the discrete label provided in the input graph and instead learn a set of latent binary attributes to describe the graph structure. Parameters for the latent attributes were learned using MAGFIT [10]. The model is sensitive to the number of latent attributes, so we repreated the experiments with 1, 2 and 3 attributes.

As these models do not generate weighted graphs, we set the weight of the edges in the generated graphs to the mean edge weight in the input graph (25.892). This ensures that statistics such as average degree strength are not skewed by unweighted edges.

To evaluate the closeness of fit of each model, we use the following statistics:

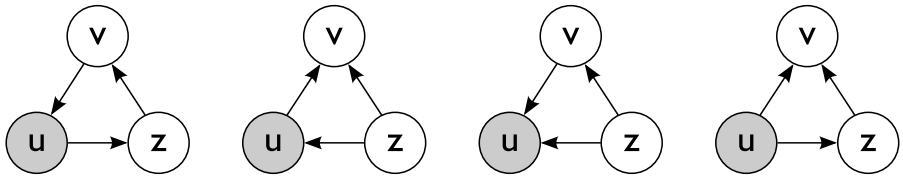
Degree Strength: For an unweighted graph, one of the most important measures is the degree distribution (the number of in-edges and out-edges of each vertex). Real-world graphs tend to have heavy-tailed power-law or log-normal degree distributions [4, 15]. For a weighted graph, we generalise the concept of vertex degree to vertex strength [8]:

$$s_u = \sum_{v \neq u} w_{uv} \tag{3}$$

For each generated graph, we plot a CDF of the in-strength, out-strength and total strength of each vertex.

Spectral Properties: We use Singular Value Decomposition (SVD) to calculate the singular values and singular vectors of the graph's adjacency matrix, which act as a signature of the most important features of the graph structure. In an unweighted graph, the adjacency matrix contains binary values, for "edge" or "no edge". In a weighted graph, the adjacency matrix contains the edge weights (with 0 indicating no edge). For SVD $U\Sigma V$, we plot CDFs of the singular values Σ and the components of the left singular vector U corresponding to the highest singular value.

Clustering Coefficients: the clustering coefficient C is an important measure of community structure. It measures the density of triangles in the graph, or the probability



(a) Cycle (b) Middleman (c) In (d) Out

Fig. 3: Triad Patterns in a Directed Graph

that two neighbours of a vertex are themselves neighbours [15]. We extend the notion of clustering coefficients to weighted, directed graphs using the equation in [8]:

$$C_u = \frac{[\mathbf{W}_u^{[\frac{1}{3}]} + (\mathbf{W}_u^T)^{[\frac{1}{3}]}]^3_{uu}}{2[d_u^{tot}(d_u^{tot}-1) - 2d_u^{\leftrightarrow}]} \tag{4}$$

where C_u is the weighted clustering coefficient for vertex u, $\mathbf{W}_u$ is the weighted adjacency matrix for u and its neighbours, $\mathbf{W}^T$ is the transpose of $\mathbf{W}$, d_u^{tot} is the total degree of a vertex (the sum of its in- and out-degrees) and $d_u^{\leftrightarrow}$ is the number of bilateral edges in u (the number of neighbours of u which have both an in-edge and an out-edge between themselves and u).

Triad Participation: Closely related to the clustering coefficient is the concept of triangle or triad participation. The number of triangles that a vertex is connected to is a measure of transitivity [15]. In a directed graph, the triangles have a different interpretation depending on the edge directions. There are four types of triangle pattern [8], as shown in Fig. 3. To generalise the concept of triad participation to weighted, directed graphs, we consider each of the four triangle types separately, and sum the total strength of the edges in each triad:

$$t_u^y = \sum_{v,z \in \mathbf{W}_u \setminus u} \mathbb{W}^y_{uvz} \tag{5}$$

where $y = \{cycle, middleman, in, out\}$ is the triangle type and $\mathbb{W}^y_{uvz}$ is calculated as shown in Fig. 3 for each triangle type y.

To give a more objective measure of the closeness of fit between the generated graphs and the input graph, we use a Kolmogorov-Smirnov (KS) test and the L2 (Euclidean) distance between the CDFs for each statistic. As the CDFs are for heavy-tailed distributions, we use the logarithmic variants of these measures [10]. The KS and L2 statistics are calculated as:

$$KS(D_1, D_2) = max_x |\log D_1(x) - \log D_2(x)| \tag{6}$$

$$L2(D_1, D_2) = \sqrt{\frac{1}{\log b - \log a} \sum_{x=a}^{b} (\log D_1(x) - \log D_2(x))^2} \tag{7}$$

where $[a, b]$ is the support of distributions D_1 and D_2.

The model that generates graphs with the lowest KS and L2 values for each of the statistics discussed above has the closest fit to the real-world graph. The results are presented in the next section.

5 Results

For each of the six models (AGWAN, MAG-R1, MAG-L1, MAG-L2, MAG-L3, ER), we generated 10 random graphs and calculated statistics for each. The plots of the averaged CDFs of the 10 graphs for each model are shown in Fig.s 4–7. The closeness of fit of each CDF (KS and L2 statistics) are shown in Tables 1–2.

Vertex Strength (Fig. 4): AGWAN successfully models the in-strength and out-strength of the input graph with high accuracy, significantly better than any of the other approaches. AGWAN successfully predicts the heavy-tailed vertex strength distribution (small number of high-strength vertices and large number of low-strength vertices).

Spectral Properties (Fig. 5): Fig. 5a shows that while AGWAN's singular values are not absolutely the closest to the values from the real graph, the shape of the distribution follows the real graph more closely than any of the other approaches. Fig. 5b shows that the components of the primary left singular vector map very closely to the real graph. Thus the spectral properties of the AGWAN graphs are very realistic, while the other models have singular vectors not much closer than random.

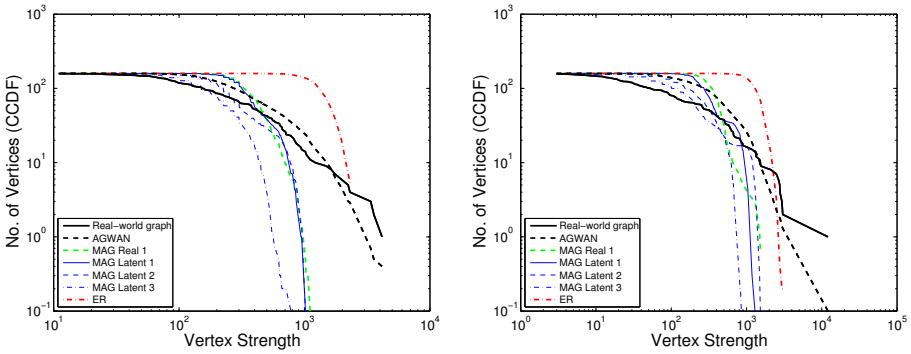
Clustering Coefficients (Fig. 6): The clustering coefficients map very closely to those from MAG-R1, outperforming it by only a small margin. Both approaches are outperformed by MAG-L1 for the In-Coefficient and MAG-L2 for the Out-Coefficient. It is interesting that there is no clear winner overall for clustering properties.

Triad Participation (Fig. 7): Triad participation is closely related to clustering, so it is not suprising to find similar results: the results for AGWAN and MAG-R1 are very similar, but MAG-L2 appears to model the real-world triad participation most closely.

Our results demonstrate that AGWAN produces an accurate model of the properties of a weighted real-world graph, significantly outperforming the unweighted approach (MAG-R1) across the statistics measured. It is interesting to note that for vertex strength and spectral qualities, AGWAN performs much better than the MAG latent approaches, whereas for clustering and triad participation, the latent approaches perform better. It is likely that the attributes in MAG-L2 and MAG-L3 are modelling different aspects of the graph structure (*e.g.* homophily and core-periphery). One of the findings in [10] was that "Simplified MAG" (where all attributes are the same) could not model the clustering property, implying that clustering cannot be accurately modelled using a single attribute. We propose to extend our model to more than one attribute as outlined in §3.3 to investigate whether this produces a more accurate model of clustering.

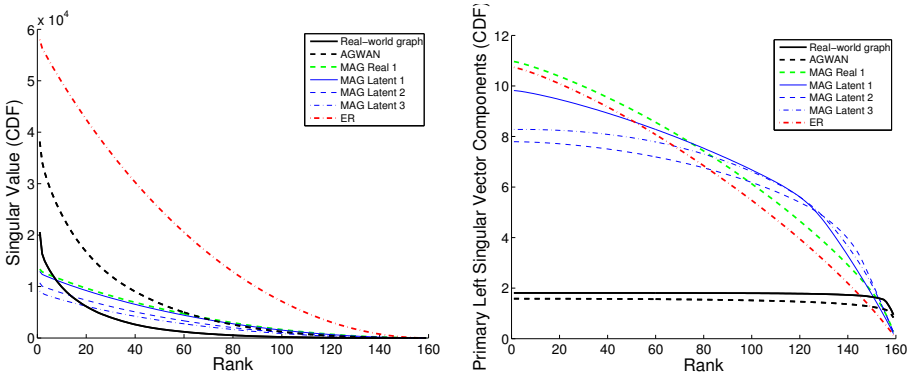
6 Conclusions

We presented AGWAN, a model for random graphs with discrete labels and weighted edges. We proposed a fitting algorithm to learn a model of graph edge weights from real-world data, and a generative algorithm to generate random graphs with similar characteristics to the real-world graph. We measured the closeness of fit of our generated graphs to the input graph over a range of graph statistics, and compared our



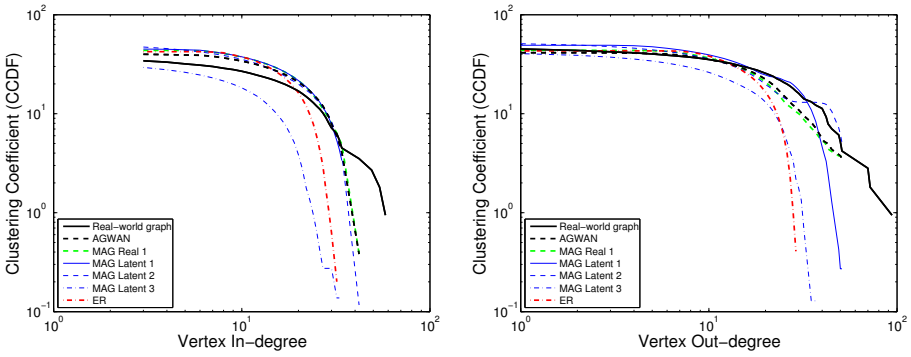
(a) In-Strength (b) Out-Strength

Fig. 4: Vertex Strength Distribution



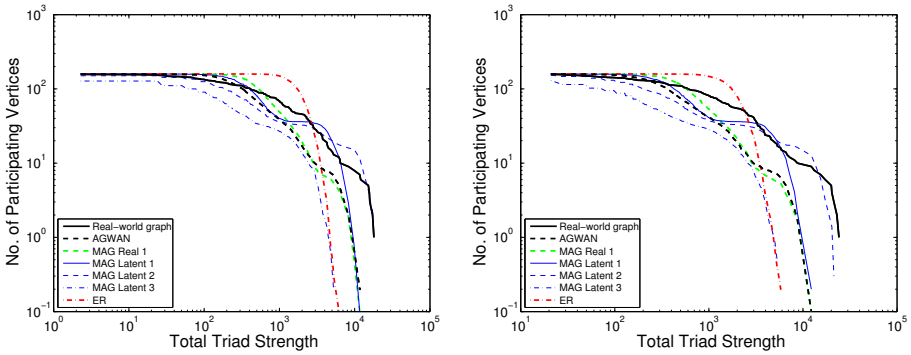
(a) Singular Values (b) Primary Left Singular Vector

Fig. 5: Spectral Properties



(a) In-edges (b) Out-edges

Fig. 6: Clustering Coefficients



(a) Cycles (b) Middlemen

(c) Ins (d) Outs

Fig. 7: Triad Participation

	AGWAN	MAG-R1	MAG-L1	MAG-L2	MAG-L3	ER
In-Strength	**1.455**	4.700	4.942	4.942	5.438	2.469
Out-Strength	**2.303**	2.659	4.942	4.605	5.247	2.708
Singular Values	34.894	35.752	35.838	34.666	**30.049**	37.235
Singular Vector	**0.282**	1.801	1.691	1.460	1.520	1.915
Clustering Coefficient (In)	2.220	2.208	**0.348**	3.406	3.821	3.444
Clustering Coefficient (Out)	0.702	0.769	2.956	**0.411**	4.628	3.728
Triad Participation (Cycles)	3.555	4.248	4.248	**0.698**	4.174	4.787
Triad Participation (Middlemen)	4.500	4.500	3.807	**2.303**	3.584	4.382
Triad Participation (Ins)	**2.436**	4.500	2.457	2.996	5.075	4.700
Triad Participation (Outs)	4.248	4.094	2.436	**0.729**	4.443	4.382

Table 1: Kolmogorov-Smirnoff statistic for CDFs in Fig.s 4–7

	AGWAN	MAG-R1	MAG-L1	MAG-L2	MAG-L3	ER
In-Strength	**1.816**	4.912	4.015	4.744	10.452	5.679
Out-Strength	**2.117**	3.534	3.252	3.420	5.188	5.100
Singular Values	18.360	19.546	19.405	**17.931**	19.262	25.044
Singular Vector	**0.988**	7.587	7.609	6.973	7.273	7.316
Clustering Coefficient (In)	1.528	1.607	**0.958**	2.226	7.175	3.528
Clustering Coefficient (Out)	1.002	1.191	2.802	**0.690**	4.716	3.145
Triad Participation (Cycles)	3.101	3.000	2.053	**1.625**	5.180	3.823
Triad Participation (Middlemen)	4.207	4.178	**2.406**	2.465	6.592	5.144
Triad Participation (Ins)	4.332	4.826	**2.166**	2.524	8.922	4.630
Triad Participation (Outs)	3.203	3.295	1.819	**1.791**	4.864	3.727

Table 2: L2 statistic for CDFs in Fig.s 4–7

approach to the state-of-the-art in random graph generative algorithms. AGWAN was found to model vertex strength distributions and singular vectors much more closely than the other approaches.

The measures of clustering and triad participation are much better than random and better than MAG with real attributes, but not as good as MAG with two latent attributes. This suggests that edge weights make some contribution to clustering, but clustering cannot be accurately modelled with a single vertex label. The MAG latent approach can assign different structural processes (homophily, core-periphery, *etc.*) to each label, and the clustering property arises from the combination of labels. We propose to extend AGWAN to multiple vertex labels to investigate the effect on clustering.

As discussed in §3.3, MAG's method of combining multiple vertex attributes is unsatisfactory when applied to real attributes, due to the assumption of independence and the limit on the number of attributes which can be modelled. We have proposed a future line of research based on a weighted summation of the GMM for each edge. The fitting algorithm would need to regularise the individual contributions of each edge to take account of dependencies. The complexity of modelling the full joint distribution could be reduced with an approach based on Markov Random Fields or Factor Graphs.

References

1. Akoglu, L., McGlohon, M., Faloutsos, C.: OddBall: Spotting anomalies in weighted graphs. In: PAKDD 2010, Hyderabad, India (2010)
2. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
3. Blei, D.M., Jordan, M.I.: Variational inference for dirichlet process mixtures. Bayesian Analysis 1, 121–144 (2005)
4. Chakrabarti, D., Faloutsos, C.: Graph Mining: Laws, Tools, and Case Studies. Synthesis Lectures on Data Mining and Knowledge Discovery, Morgan & Claypool Publishers (2012)
5. Davis, M., Liu, W., Miller, P.: Finding the most descriptive substructures in graphs with discrete and numeric labels. In: NFMCP, pp. 138–154. Springer (2013)
6. Eichinger, F., Huber, M., Böhm, K.: On the usefulness of weight-based constraints in frequent subgraph mining. In: ICAI 2010. pp. 65–78. BCS SGAI (Dec 2010)
7. Erdős, P., Rényi, A.: On the evolution of random graphs. Publication of the Mathematical Institute of the Hungarian Academy of Sciences 5, 17–61 (1960)
8. Fagiolo, G.: Clustering in complex directed networks. Physical Review E 76(2) (Aug 2007)
9. Figueiredo, M.A., Jain, A.: Unsupervised learning of finite mixture models. Pattern Analysis and Machine Intelligence, IEEE Transactions on 24(3), 381–396 (2002)
10. Kim, M., Leskovec, J.: Modeling social networks with node attributes using the Multiplicative Attribute Graph model. In: UAI 2011, Barcelona, Spain. pp. 400–409 (2011)
11. Kim, M., Leskovec, J.: Multiplicative Attribute Graph model of real-world networks. Internet Mathematics 8(1–2), 113–160 (2012)
12. Kurihara, K., Welling, M., Vlassis, N.: Accelerated variational dirichlet process mixtures. In: NIPS (2006)
13. Leskovec, J., Chakrabarti, D., Kleinberg, J.M., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: An approach to modeling networks. JMLR 11, 985–1042 (2010)
14. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. ACM Trans. Knowl. Discov. Data 1(1) (Mar 2007)
15. Newman, M.: Networks: An Introduction. OUP, New York, NY, USA (2010)

Feature extraction over multiple representations for time series classification

Dominique Gay, Romain Guigourès, Marc Boullé, and Fabrice Clérot

Orange Labs
2, avenue Pierre Marzin, F-22307 Lannion Cedex, France
firstname.name@orange.com

Abstract. We suggest a simple yet effective and parameter-free feature construction process for time series classification. Our process is decomposed in three steps: *(i)* we transform original data into several simple representations; *(ii)* on each representation, we apply a coclustering method; *(iii)* we use coclustering results to build new features for time series. It results in a new transactional (i.e. object-attribute oriented) data set, made of time series identifiers described by features related to the various generated representations. We show that a Selective Naive Bayes classifier on this new data set is highly competitive when compared with state-of-the-art times series classification methods while highlighting interpretable and class relevant patterns.

1 Introduction

Time series classification (TSC) has been intensively studied in the past years. The goal is to predict the class of an object (a time series or a curve) $\tau_i = \langle (t_1, x_1), (t_2, x_2), \ldots, (t_{m_i}, x_{m_i}) \rangle$ (where $x_k, (k = 1..m_i)$ is the value of the series at time t_k), given a set of labeled training time series. TSC problems differ from traditional classification problems since there is a time dependence between the variables ; in other terms, the order of the variables is crucial in learning an accurate predictive model. The increasing interest in TSC is certainly due to the wide variety of applications: from e.g., medical diagnosis (like classification of patient electrocardiograms) to the maintenance of industrial machinery. Other domains, where data might be time series, are also concerned: finance, meteorology, signal processing, computer network traffic, ... The diversity of applications has given rise to numerous approaches (see Section 4 for detailed related work). However, most efforts of the community have been devoted to the following three-step learning process: *(i)* choosing a new data representation, *(ii)* choosing a similarity measure (or a distance) to compare two time series and *(iii)* using the Nearest Neighbor (NN) algorithm as classifier on the chosen representation, using the chosen measure. Wang et al. [17] offer a survey of the various data representations and distances found in the literature and an extensive experimental study using the NN classifier. They conclude that NN classifier coupled with

Euclidean distance (ED) or Dynamic Time Warping (DTW) show the highest predictive performance for TSC problems using the original time domain. Recently, Bagnall et al. [2] experimentally show that the performance of classifiers significantly increases when changing data representation (compared with original temporal domain) ; thus, for a given classifier, there is a high variance of performance depending on the data transformation at use. To alleviate this problem, an ensemble method TSC-ENSEMBLE [2] based on three data representations (plus the original data) and NN algorithm is suggested. The experimental results demonstrate the importance of representations in TSC problems and show that a simple ensemble method based on several data representations provides highly competitive predictive performance. However, with the good performance of NN-based approaches also come the drawbacks of lazy learners: i.e., there is no proper training phase, therefore the training set has to be entirely stored and all the computation time is postponed until deployment phase. Another weakness of the NN approaches is the lack of interpretability; indeed NN only indicates the nearest series w.r.t. the used similarity measure.

The method we suggest takes the pros and leaves the cons of the methods listed above: we come back to the *eager*[1] paradigm, benefit from the combination of multiple representations, build and select valuable features from multiple representations. More precisely, in this paper, we suggest a parameter-free process for constructing valuable features over multiple representations for TSC problems. Our contribution is thus essentially methodological. The next section motivates and describes the three steps of our unsupervised process: *(i)* transformation of original data into several new data representations; *(ii)* coclustering on various data representations ; *(iii)* the exploitation of coclustering results for the construction of new features for the data. The output of the process is then a traditional data set (i.e. labeled objects described by attributes) ready for supervised feature selection and classification. We report the experimental validation of our approach in section 3 and discuss further related work in Section 4 before concluding.

2 Feature construction process

Notations. In TSC problems, we define a time serie as a pair (τ_i, y_i) where τ_i is a set of ordered observations $\tau_i = \langle (t_1, x_1), (t_2, x_2), \ldots, (t_{m_i}, x_{m_i}) \rangle$ of length m_i and y_i a class value. A time series data set is defined as a set of pairs $D = \{(\tau_1, y_1), \ldots, (\tau_n, y_n)\}$, where each time series may have a different number of observations (i.e. with different length). Notice that the time series of a data set may also have different values for $t_k, (k = 1..m_i)$. The goal is to learn a classifier from D to predict the class of new incoming time series $\tau_{n+1}, \tau_{n+2}, \ldots$ To achieve this goal, we suggest the feature construction process summarized as follows:

[1] In contrast to lazy learning, eager learning has an explicit training phase and generally deploys faster.

1. We transform original data into multiple data representations.
2. We process a coclustering technique on each representation.
3. We build a set of features from each coclustering result and obtain a new data set gathering the various sets of features.

The new data set is thus object-attribute oriented and ready for supervised classification phase. Since our main contribution is methodological, we will take some time to motivate each step of the process, and when necessary, to make the paper self-contained, we will recall the main principles of the tools used in each step.

2.1 Transformations & Representations

Numerous data transformation methods for time series has been suggested in the literature: e.g., polynomial, symbolic, spectral or wavelet transformations, ... (see [17] for a well-structured survey on experienced data representations).

The underlying idea of using data transformation is that transformed data might contain class-characteristic pattern that are easily detectable (i.e. patterns unreachable in the original time domain). The following example illustrates and confirms the relevance of using representations, and highlights simple interpretable features that might arise from data representations.

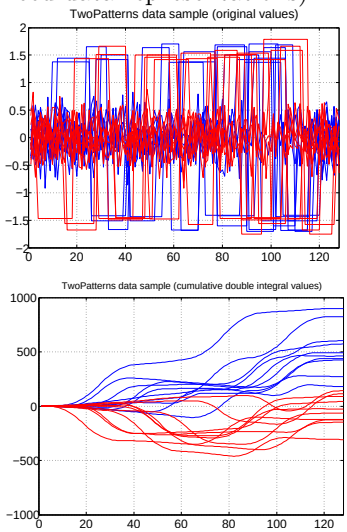


Fig. 1. An extract from the TwoPatterns data set (20 series, 2 classes) in its original representation and in double cumulative integral representations

Motivating example. Graphs from figure 1 confirm the relevance of changing data representation: indeed, from original data (a) it is uneasy to separate the two classes (blue/red) whereas simple transformation, like cumulative double integral (b) facilitate class discrimination. For example, after computing the cumulative double integral transformation, we see that curves with some values above 100 are blue and curves with some values below -100 are red. On this tiny example (extracted from the TwoPatterns data set from UCR repository [10]), a simple transformation and two interpretable features are enough to characterize the two classes of curves.

To illustrate and instantiate our process, we use the original representation and we pick six representations among the numerous ones existing in the literature.

Derivatives : DV et DDV We use derivatives and double derivatives of original time series (computed between time t et $t-1$). These transformations allow us to represent the local evolution (i.e., increasing/decreasing, acceleration/deceleration) of the series.

Cumulative integrals : IV et IIV We also use simple and double cumulative

integrals of the series, computed using the trapeze method. These transformations allow us to represent the global (cumulated) evolution of the series.
Power Spectrum : PS. A time series can be decomposed in a linear combination of sines and cosines with various amplitudes and frequencies. This decomposition is known as the Fourier transform. And, the Power Spectrum is $PS(\tau_i) = \langle (f_1, a_1), \ldots, (f_{m_i}, a_{m_i}) \rangle$, where f_k represent the frequency domain and a_k the power of the signal (i.e. the sum of the Fourier coefficients squared). This transformation is commonly used in signal processing and plunges the original series into the frequency domain.
Auto-correlation function : ACF The transformation by auto-correlation (ACF) is : $\tau_{i\rho} = \langle (t_1, \rho_1), \ldots (t_{m_i}, \rho_{m_i}) \rangle$ where

$$\rho_k = \frac{\sum_{j=1}^{j=m_i-k} (x_j - \bar{x}) \cdot (x_{j+k} - \bar{x})}{m \cdot s^2}$$

and where $\bar{x}$ and s^2 are the mean and variance of the original series. ACF transformation describes the correlation between values of the signal at different times and thus allow us to represent auto-correlation structures like repeating patterns in the time series.

Thus, for a given time series data set D_{orig}, we build six new data representations: D_{DV}, D_{DDV}, D_{IV}, D_{IIV}, D_{PS} and D_{ACF} depending on the transformation used. In the following, for the sake of generality, an object from one of these representations will be called "curve" instead of time series since D_{PS} does not use the time domain.

2.2 Coclustering

In classification problems (also in TSC), there might exist intra-class variance, i.e. the variations between objects of the same class might be numerous and of various aspects. Using clustering as a pre-processing step to supervised classification is not new and is a solution to deal with intra-class variance. The idea is to pre-process the data set by grouping together similar objects and to highlight local patterns that might be class-discriminant: e.g., Vilalta et al. [16] suggest a pre-processing step by supervised (per-class) clustering using Expectation Maximization to enhance the predictive performance of Naive Bayes classifier. In order to be able to derive interesting features, we will use an unsupervised coclustering technique as described in the following.

A curve can be seen as a set of points (X, Y), described by their abscissa and ordinate values. A set of curves is then also a set of points (C_{id}, X, Y) where C_{id} is the curve identifier. This tridimensional representation (one categorical variable and two numerical variables) of a curve data set is needed to apply coclustering methods. Indeed, the goal is to partition the categorical variable and to discretize the numerical variables in order to obtain clusters of curves and intervals for X and Y. The result is a tridimensional grid whose cells are defined by a group of curves, an interval for X and an interval for Y.
For that purpose, we use the coclustering method KHC [6] (Khiops Coclustering). Originally designed for clustering functional data, it is also suitable for

the particular case of curve data as defined above and it is directly applicable for our pre-processing step. KHC method is based on a piecewise constant non-parametric density estimation and instantiates the generic MODL approach [4] (Minimum Optimized Description Length) – which is similar to a Bayesian Maximum A Posteriori (MAP) approach. The optimal model M, i.e. the optimal grid, is obtained by optimization of a Bayesian criterion, called *cost*. The *cost* criterion bets on a trade-off between the accuracy and the robustness of the model and is defined as follows:

$$cost(M) = -\log(\underbrace{p(M \mid D)}_{\text{posterior}}) = -\log(\underbrace{p(M)}_{\text{prior}} \times \underbrace{p(D \mid M)}_{\text{likelihood}})$$

Using a hierarchical prior (on the parameters of a data grid model) that is uniform at each stage of the hierarchy, we obtain an analytic expression for the *cost* criterion:

$$cost(M) = \log n + 2\log N + \log B(n, k_C) \tag{1}$$

$$+ \log \binom{N+k-1}{k-1} + \sum_{i_C=1}^{k_C} \log \binom{N_{i_C} + n_{i_C} - 1}{n_{i_C} - 1} \tag{2}$$

$$+ \log N! - \sum_{i_C=1}^{k_C} \sum_{j_X=1}^{k_X} \sum_{j_Y=1}^{k_Y} \log N_{i_C j_X j_Y}! \tag{3}$$

$$+ \sum_{i_C=1}^{k_C} \log N_{i_C}! - \sum_{i=1}^{n} \log N_i! + \sum_{j_X=1}^{k_X} \log N_{j_X}! + \sum_{j_Y=1}^{k_Y} \log N_{j_Y}! \tag{4}$$

where n is the number of curves, N the number of points, k_C (resp. k_X, k_Y) is the number of clusters of curves (resp. the number of intervals for X and Y), k the number of cells of the data grid, n_{i_C} the number of curves in cluster i_C, N_i the number of points for curve i and N_{i_C} (resp. N_{j_X}, N_{j_Y}, $N_{i_C j_X j_Y}$) is the cumulated number of points for curves of cluster i_C (resp. for interval j_X of X, interval j_Y of Y, for cell (i_C, j_X, j_Y) of the data grid. Notice that $B(n, k_C)$ is the number of divisions of n elements into k subsets. The two first lines stand for the prior and the two last lines relates to the likelihood of the model. Intuitively, low *cost* means high probability ($p(M \mid D)$) that the model M arises from the data D. From an information theory point of view, according to [15], the negative logarithms of probabilities may be interpreted as code length. Thus, the *cost* criterion may also be interpreted as the code length of the grid model plus the code length of data D given the model M, according to the Minimum Description Length principle (MDL [9]). Here, low *cost* means high compression of the data using the model M.

The *cost* criterion is optimized using a greedy bottom-up strategy, *(i)* starting with the finest grained model, *(ii)* considering all merges between adjacent clusters or intervals, for the curve and dimension variables, and *(iii)* performs the best merge if the criterion decreases after the merge. The process loops until no further merge improves the criterion. The obtained grid constitutes a non-parametric estimator of the joint density of the curves and the dimensions of points.

KHC is parameter-free, robust (avoids over-fitting), handles large curve data sets with several millions of data points and its time complexity is $\Theta(N\sqrt{N}\log N)$ (sub-quadratic) where N is the number of data points: thus, KHC meets our problem needs (for full details, see [6]).

2.3 Feature construction

Feature construction for TSC problems [13] aims at capturing class-relevant properties for describing time series. The generated features goes from simple ones like minimum, maximum, mean, standard deviation of time series to more complex ones like e.g., coefficients of spectral decompositions [12]. The main advantage of feature-based approaches is the final transactional (or vector) representation of the data which is suitable for conventional classifiers like Naive Bayes or decision trees. In our process, we generate features from coclustering results as follows.

For each coclustering result obtained with KHC on a data representation (D_{orig}, D_{DV}, D_{DDV}, D_{IV}, D_{IIV}, D_{PS}, D_{ACF}), we create a set of new features: $\mathcal{F}_{orig}, \mathcal{F}_{DV}, \mathcal{F}_{DDV}, \mathcal{F}_{IV}, \mathcal{F}_{IIV}, \mathcal{F}_{PS}, \mathcal{F}_{ACF}$. The new features are the descriptive attributes of the new data set whose objects are curves.

Let D_{rep} be one of the seven representations described above. Let $M_{rep} = KHC(D_{rep})$ be the tridimensional optimal grid obtained by coclustering with KHC on D_{rep}. We denote k_C the number of clusters of M_{rep} and k_Y the number of intervals of M_{rep} for dimension Y. We then create similarity-based features and histogram features.

Similarity-based features
Considering the good performance of (dis)similarity-based approaches (e.g., ED-NN and DTW-NN), we define a dissimilarity index based on the *cost* criterion.

Definition 1 (Dissimilarity index). *The dissimilarity between a curve τ_i and a cluster c_j of the optimal grid M_{rep} is defined as:*

$$d(\tau_i, c_j) = cost(M_{rep|\tau_i \cup c_j}) - cost(M_{rep})$$

i.e., the difference of cost between the optimal model M_{rep} and the model $M_{rep|\tau_i \cup c_j}$ (the optimal grid in which we add the curve τ_i to the cluster of curves c_j).

Intuitively, d measures the perturbation brought by the integration of a curve into a cluster of curves of the optimal grid (i.e. according to the *cost* criterion used for grid optimization). In terms of code length, if a curve τ_i is similar to the curves of cluster c_j, the total code length of the data is not much different from the total code length of the data plus τ_i. Thus, small values of $d(\tau_i, c_j)$ indicate that τ_i is similar to the curves of c_j whereas high values of d ($d(\tau_i, c_j) \gg 0$) mean that τ_i does not look like the curves of c_j.
According to the dissimilarity index d, we generate the following features:

– k_C numerical features (one for each cluster c_j of curves of M_{rep}). The value for a curve τ_i is the difference $d(\tau_i, c_j)$. Thus, for a given curve τ_i, these features tell how τ_i is similar to the clusters of curves of the optimal grid (according to d).
– One categorical feature indicating the index j of the cluster of curves that is the closest to a curve τ_i according to the dissimilarity d defined above (i.e., $\arg\min_j d(\tau_i, c_j)$).

Histogram features
Taking up the idea of interpretable features (see motivating example and fig.1), we also generate the following features:

– k_Y numerical features (one for each interval i_Y of Y from M_{rep}) whose value for a curve τ_i is the number of points of τ_i in interval i_Y.

These histogram features quantify the presence of a curve in intervals of Y obtained in the coclustering step.

For a given curve τ_i, we now have the following informations provided by the new features (for each representation): *(i)* the dissimilarity values between τ_i and all the clusters of curves, *(ii)* the index of the closest cluster of curves and *(iii)* the number of points of τ_i in each interval of Y.

2.4 Supervised classification algorithm

We saw that our feature construction process may generate hundreds of new features for each representation. The whole set of features $\mathcal{F}_{tot}$ for our new data set may contain thousands of attributes. Therefore, the classifier at the end of our process has to be capable of handling a large number of attributes but also selecting the relevant attributes for the classification task. At this stage, we could use conventional classifiers like decision trees or SVM. However, we choose the Selective Naive Bayes classifier (SNB) that meets all the needs, is parameter-free and outperforms classical Naive Bayes [5]. Notice that SNB exploits pre-processing techniques that discretize numerical variables, group values of categorical variables, weight and select features w.r.t. class-relevance by using robust conditional density estimators and following the MODL approach (see [3], [4]). Thus, the generated features benefit from these pre-processing techniques and preserve a potential of interpretability; we lead specific experiments in the next section to support this claim. Moreover, SNB is parameter-free, so is the whole feature construction process. Its time complexity is $\Theta(KN\log(KN))$, where N is the number of objects and K the number of features.

3 Experimental validation

The implementation of the classification process is based on existing tools (KHC for coclustering and SNB for supervised classification[2]). Connections between

[2] KHC and SNB are both available at `http://www.khiops.com`

the tools to handle the whole process, named MODL-TSC, are scripted using MATLAB. The experiments are led to discuss the following questions:

$\mathcal{Q}_1$ Is MODL-TSC comparable with competitive contenders of the state-of-the-art in terms of accuracy?
$\mathcal{Q}_2$ MODL-TSC employs and combines several representations. Are they all useful? Do they all bring the same impact?
$\mathcal{Q}_3$ What kind of data insight do we gain using the coclustering-based features?

3.1 Protocol

We experiment our process on 51 time series data sets: 42 data sets are from UCR [10] and 9 new data sets introduced in [2]. The benchmark data sets offer a wide variety in terms of application domains, number of series, length of series and number of class values. We lead experiments in a predefined train-test setting for each data set (see [10]). We compare the predictive performance of our process, called MODL-TSC, with a baseline and three of the most effective alternative approaches:

- ED-NN: the Nearest Neighbor classifier using the Euclidean distance. This approach is considered as a baseline.
- DTW-NN: the Nearest Neighbor classifier using the elastic distance Dynamic Time Warping, considered as hard to beat in the literature (see [18])
- TSC-ENSEMBLE [2] exploits multiple representations via an ensemble method and the NN algorithm. Its performance is comparable to DTW-NN
- FAST-SHAPELETS [14] mines shapelets (i.e., class relevant time series subsequences) that might be embedded in e.g., a decision tree

3.2 Results

Due to space limitations, only average accuracy are reported. For full details about predictive performance and running time results, see [1].

Comparisons with state-of-the-art. Firstly, global results (mean accuracy, number of wins and mean rank) show that MODL-TSC is very competitive compared to state-of-the-art methods (see table 1). Although ED-NN seems to perform worse than other contenders, the Friedman test (at significance level $\alpha = 0.05$) does not allow us to reject the null hypothesis, i.e., considering the experiments on these 51 benchmark data sets, there is no classifier singled out. We also run Wilcoxon's sign rank test for pairwise comparisons (also with $\alpha = 0.05$): it appears that MODL-TSC (as well as DTW-NN and TSC-ENS) significantly performs better than ED-NN but other pairwise comparisons conclude that there is no significant difference of performance between MODL-TSC and DTW-NN or TSC-ENS. Table 2 reports the comparison between MODL-TSC and FAST-SHAPELETS over 45 data sets (since accuracy results for FAST-SHAPELETS are available for these data, see [14]). Here Wilcoxon's sign rank test for pairwise comparisons indicates that MODL-TSC performs significantly better than FAST-SHAPELETS. This

global view of performance results confirms that MODL-TSC is very competitive compared to the most effective contenders of the state-of-the-art.

Data	DTW-NN	ED-NN	TSC-ENS	MODL-TSC
Mean Acc	78.06	73.89	77.45	77.44
#wins	21	2	10	20
MODL-TSC wins vs.	25	34	29	-
Average rank	2.2549	3.1372	2.2745	2.2941

Table 1. Comparisons of accuracy results over 51 data sets for MODL-TSC, DTW-NN, ED-NN, TSC-ENSEMBLE.

Data	MODL-TSC	SHAPELETS
Mean Acc	76.68	72.81
#Wins	32	13

Table 2. Comparisons of accuracy results over 45 data sets for MODL-TSC and FAST-SHAPELETS.

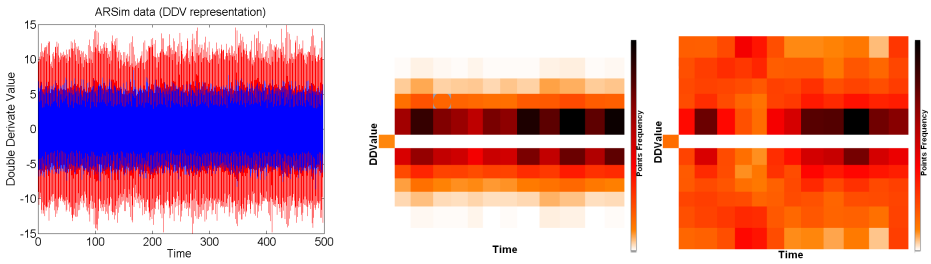
Secondly, we observe the remarkable performance of MODL-TSC on ARSim, ElectricDevices, FordA and OSULeaf data. On these data, we outperform DTW-NN and TSC-ENSEMBLE: the difference of test accuracy is at least 10. Here, the added-value of the data representations (i.e., the new features) is at work. TSC-ENSEMBLE (exploiting only three representations) and DTW-NN (working in time domain) obtain only poor accuracy results. Conversely the performance of MODL-TSC is very low on Coffee, DiatomSizeReduction, ECGFiveDays and OliveOil data. The difference of test accuracy (about 10 compared with DTW-NN and TSC-ENSEMBLE) is now to our disadvantage. We think that this poor performance might due to one reason: the training set size of these data sets is very small (less than 30 of curves) and could be insufficient for either learning relevant coclusters or learning a predictive model without over-fitting. Indeed, e.g., for OliveOil data, there are only 30 training curves (for 4 classes), no cluster of curves is found by KHC whatever the representation.

Added-value of the representations. In table 3, we also report accuracy results of SINGLE-MODL-TSC using only one representation.

We observe that using one single representation provide poor average accuracy results. Almost always, MODL-TSC using several representations outperforms SINGLE-MODL-TSC on $\mathcal{F}_{orig}$ (resp. $\mathcal{F}_{DV}$, $\mathcal{F}_{DDV}$, $\mathcal{F}_{IV}$, $\mathcal{F}_{IIV}$, $\mathcal{F}_{PS}$, $\mathcal{F}_{ACF}$). In some cases (e.g., ItalyPowerDemand, MALLAT or MedicalImages), the good performance of MODL-TSC can be attributed to the combination of several representations. Indeed, the gap of test accuracy between any SINGLE-MODL-TSC and MODL-TSC is about 10; thus the combination of features coming from different views of the data improves accuracy results. In other cases (e.g., ARSim or wafer), the good performance seems to be due to only one (or at most two) representation while the other representations are ignored. As an example, for ARSim data, the DDV representation is the most relevant. KHC obtains 43 clusters of curves and 12 intervals for Y_{DDV}. Most of the clusters are almost pure (only one class of curves per cluster). Moreover, as we can see in figure 2(b) and (c), the number of points in intervals generated by KHC above 6 and below -6 are class-discriminant since curves of class 1 almost never have points in these regions.

	Mean Acc	MODL-TSC wins
MODL-TSC	77.44	-
$\mathcal{F}_{orig}$	65.98	49
$\mathcal{F}_{DV}$	61.20	47
$\mathcal{F}_{DDV}$	55.30	49
$\mathcal{F}_{IV}$	59.26	50
$\mathcal{F}_{IIV}$	53.48	50
$\mathcal{F}_{PS}$	53.27	50
$\mathcal{F}_{ACF}$	57.54	49

Table 3. Comparisons of accuracy results for MODL-TSC and SINGLE-MODL-TSC using each single representation.



(a) (b) (c)

Fig. 2. ARSim data: (a) double derivate representation, class 1 is blue and class 2 is red. (b) An example of cluster whose curves are mostly of class 1. (c) an example of cluster whose curves are mostly of class 2. The frequency of points are represented for each cell in Time vs. DDValue axis. The stronger the color, the more frequent are the points in a cell.

These experiments recall the very importance of representations in TSC problems and particularly in our feature construction process. Even the simple representations we chose to illustrate our process show good predictive performance. Depending on the application, we may still hope some improvement in performance if we could rely on expert domain knowledge to select relevant representations to use in our generic process.

3.3 Interpretation: an example

If we consider the cumulative integral (IV) representation of TwoPatterns data, the optimal grid obtained by KHC is made of 224 clusters of curves, 11 intervals for X and 9 intervals for Y_{IV}. According to the MODL pre-processing techniques, among all the attributes generated from all representations, the two most relevant attributes are from the IV representation:

1. v_1, the number of points in interval $I_{Y_{IV}} =]-\infty; -3.9082]$
2. v_2, the index of the closest cluster

In the supervised learning step, the discretization for v_1 and the value grouping for v_2 provide the following contingency tables represented as histograms (see figure 3). We observe (figure 3(a)) that the number of points p of a curve in interval $I_{Y_{IV}}$ (i.e. the number of points with value less than -3.9082) is class relevant. Indeed, in the learning phase, curves such that $p \leq 7$ are of class c_1; when $p > 29$ (about 23% of the points of the curve), curves are mostly of class c_4 and when $7 < p \leq 12$ they are mostly of class c_3. This type of feature is similar to the ones in the motivating example and figure 1: for a given representation, some regions of y-axis (delimited by intervals) will be class-discriminant and the number of points of an incoming curve in this interval will also be class-discriminant.
In figure 3(b), we firstly see that, for variable v_2 ("index of the closest cluster"), MODL pre-processing by supervised value grouping provide 4 groups: G_1, (resp. G_2, G_3 et G_4) made of 56, (resp. 53, 53 et 62) indexes of clusters that are mostly of class c_4 (resp. c_3, c_2, c_1). The attribute v_2 is then class-relevant. Indeed, for example, if j is the index of cluster, that is the closest to a curve τ_i, and belongs

to G_2 (i.e. $j \in G_2$), then τ_i is considered very similar to curves of class c_3. Moreover, the variable "index of the closest cluster" is an indicator of the relevance of the representation in our process for the current TSC problem. In this example, attribute v_2 alone, is enough to characterize 95% of the data, therefore, IV data representation is very relevant for characterizing the classes of TwoPatterns data. Conversely, for the original representation (D_V), the optimal grid obtained with KHC is made of 255 clusters of curves but MODL pre-processing indicates that the variable "index of the closest cluster" is not relevant to characterize the classes of TwoPatterns; as a consequence, SINGLE-MODL-TSC on $\mathcal{F}_{orig}$ shows bad test accuracy results.

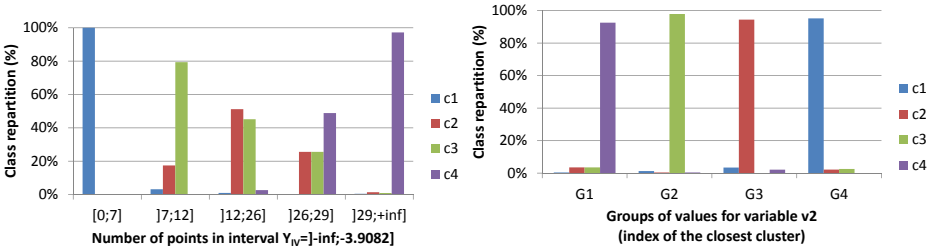


Fig. 3. Histogram representation of class repartition for discretization of variable v_1 and value grouping of variable v_2.

4 Related work

In TSC problems, DTW-NN is recognized by the community as a hard-to-beat baseline and it is confirmed by our experiments. However, there exist alternative approaches: besides the numerous similarity measures coupled with Nearest Neighbor algorithm [17], for the sake of interpretability, feature-based approaches have also been intensively studied. Feature-based approaches for TSC aim at extracting class-relevant characteristics of series so that a conventional classifier can be used. A wide variety of features has been studied: e.g., global, trends, symbolic, intervals, distance-based, features coming from spectral transforms [12] or a combination of several types of features [7].

Shapelet-based approaches, a subtopic of feature-based approaches, have drawn much attention in recent years. Shapelets are time series subsequences that are representative of a class. First approaches have embedded extracted shapelets in a decision tree [8, 20], others in a simple rule-based classifier [19], while very recently, Lines et al. [11] have designed a shapelet-based transform.

Our approach generates similarity-based features and histogram features over multiple representations; the former allows us to reach predictive performance comparable to the best similarity-based NN classifiers, with the latter we gain some insight in the data. The closest works are that of Eruhimov et al. [7] who employs a generation of high dimensional feature space and the inspiring work of Bagnall et al. [2] who establish competitive predictive performance by combining multiple representations in an ensemble classifier.

5 Conclusion & Perspectives

We have suggested MODL-TSC, a simple yet effective and generic feature construction process for time series classification problems (TSC). Our process is parameter-free, easy to use and the generated features offer a high potential of interpretation. The time complexity of our process is sub-quadratic, thus time-efficient. Experimental results show that the performance of MODL-TSC is highly competitive and comparable with two of the most accurate approaches of the state-of-the-art (namely, DTW-NN and TSC-ENS).

The first results are promising and also confirm the importance of representations in TSC problems. Indeed, depending on the application domain, a particular transformation will facilitate the discovery of class relevant patterns. Moreover, the combination of multiple representations with MODL-TSC leads to highly competitive predictive performance. We have used only a few simple representations in the time, frequency and correlation domains to demonstrate that our feature construction approach is well-founded. The literature offers plenty of relevant data representations (see [17] for a wide view). Notice also that designing new representations is still a hot topic (see e.g., [11]). It gives an *unexplored* potential of improvement for MODL-TSC on data sets and applications where we are less performant.

References

1. Supporting webpage, `http://sites.google.com/site/tscfeatures/`
2. Bagnall, A., Davis, L.M., Hills, J., Lines, J.: Transformation based ensembles for time series classification. In: SDM'12. pp. 307–318 (2012)
3. Boullé, M.: A bayes optimal approach for partitioning the values of categorical attributes. Journal of Machine Learning Research 6, 1431–1452 (2005)
4. Boullé, M.: MODL: A bayes optimal discretization method for continuous attributes. Machine Learning 65(1), 131–165 (2006)
5. Boullé, M.: Compression-based averaging of selective naive Bayes classifiers. Journal of Machine Learning Research 8, 1659–1685 (2007)
6. Boullé, M.: Functional data clustering via piecewise constant nonparametric density estimation. Pattern Recognition 45(12), 4389–4401 (2012)
7. Eruhimov, V., Martyanov, V., Tuv, E.: Constructing high dimensional feature space for time series classification. In: PKDD'07. pp. 414–421 (2007)
8. Geurts, P.: Pattern extraction for time series classification. In: PKDD'01. pp. 115–127 (2001)
9. Grünwald, P.: The minimum description length principle. MIT Press (2007)
10. Keogh, E., Zhu, Q., Hu, B., Y., H., Xi, X., Wei, L., Ratanamahatana, C.A.: The UCR time series classification/clustering page (2011), `http://www.cs.ucr.edu/~eamonn/time_series_data/`
11. Lines, J., Davis, L.M., Hills, J., Bagnall, A.: A shapelet transform for time series classification. In: KDD'12. pp. 289–297 (2012)
12. Mörchen, F.: Time series feature extraction for data mining using DWT and DFT. Tech. rep., Philipps Univeristy Marburg (2003)

13. Nanopoulos, A., Alcock, R., Manolopoulos, Y.: Feature-based classification of time-series data. In: Mastorakis, N., Nikolopoulos, S.D. (eds.) Information Processing and Technology, pp. 49–61. Nova Science (2001)
14. Rakthanmanon, T., Keogh, E.: Fast shapelets: a scalable algorithm for discovering time series shapelets. In: SIAM DM'13 (2013)
15. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal (1948)
16. Vilalta, R., Rish, I.: A decomposition of classes via clustering to explain and improve naive bayes. In: ECML'03. pp. 444–455 (2003)
17. Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.: Experimental comparison of representation methods and distance measures for time series data. Data Mining and Knowledge Discovery 26(2), 275–309 (2013)
18. Xi, X., Keogh, E.J., Shelton, C.R., Wei, L., Ratanamahatana, C.A.: Fast time series classification using numerosity reduction. In: ICML'06. pp. 1033–1040 (2006)
19. Xing, Z., Pei, J., Yu, P.S., Wang, K.: Extracting interpretable features for early classification on time series. In: SDM'11. pp. 247–258 (2011)
20. Ye, L., Keogh, E.J.: Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Mining and Knowledge Discovery 22(1-2), 149–182 (2011)

Mining Frequent Partite Episodes with Partwise Constraints

Takashi Katoh, Shin-ichiro Tago, Tatsuya Asai, Hiroaki Morikawa,
Junichi Shigezumi, and Hiroya Inakoshi

Fujitsu Laboratories Ltd., Kawasaki, 211-8588, Japan
{ kato.takashi_01, s-tago, asai.tatsuya, h.morikawa, j.shigezumi,
inakoshi.hiroya } @jp.fujitsu.com

Abstract. In this paper, we study the problem of efficiently mining frequent *partite episodes* that satisfy *partwise constraints* from an input event sequence. Through our constraints, we can extract episodes related to events and their precedent-subsequent relations, on which we focus, in a short time. This improves the efficiency of data mining using trial and error processes. A partite episode of length k is of the form $P = \langle P_1, \ldots, P_k \rangle$ for sets P_i $(1 \leq i \leq k)$ of events. We call P_i a *part* of P for every $1 \leq i \leq k$. We introduce the *partwise constraints* for partite episodes P, which consists of *shape and pattern constraints*. A shape constraint specifies the size of each part of P and the length of P. A pattern constraint specifies subsets of each part of P. We then present a backtracking algorithm that finds all of the frequent partite episodes satisfying a partwise constraint from an input event sequence. By theoretical analysis, we show that the algorithm runs in output polynomial time and polynomial space for the total input size. In the experiment, we show that our proposed algorithm is much faster than existing algorithms for mining partite episodes on artificial datasets.

1 Introduction

Episode Mining. One of the most important tasks in data mining is to discover frequent patterns from time-related data. Mannila *et al.* [6] introduced *episode mining* to discover frequent *episodes* in an event sequence. An episode is formulated as a labeled acyclic digraph in which labels corresponding to events and arcs represent a temporal precedent-subsequent relation in an event sequence.

For subclasses of episodes [6, 3], a number of efficient algorithms have been developed. In a previous work [3], we introduced the class of *partite episodes*, which are time-series patterns of the form $\langle P_1, \ldots, P_k \rangle$ for sets P_i $(1 \leq i \leq k)$ of events, which means that in an input event sequence, every event in P_{i+1} follows every event in P_i for every $1 \leq i < k$. For the class of partite episode, we presented an algorithm that finds all of the frequent episodes from an input event sequence.

A partite episode is a richer representation of temporal relationship than a subsequence, which represents just a linearly ordered relation in sequential pat-

tern mining [1, 9]. In particular, the partite episode can represent the precedent-subsequent relation between the sets of events that occur simultaneously (in any order) in a time span.

Main Problem. In the data analysis using trial and error processes, we often want to extract only episodes which include some events we focus on. For example, if we get an episode that means *buying a telescope before buying a PC*, then we will be interested in frequent patterns in the period between the two events.

In this paper, we introduce *partwise constraints* for partite episodes. A partwise constraint consists of a *shape constraint* and a *pattern constraint*. For a partite episode P, a shape constraint specifies the size of each part of P and the length of P. A pattern constraint specifies the subsets of each part of P. Through episode mining with a shape constraint that means the length of any extracted episode is at most three and both the size of the first and the third set are at most one, and a pattern constraint that means the first set includes the event *buying a PC* and the third set includes the event *buying a telescope*, we may obtain a knowledge: *some customers bought a digital camera and a photo printer after buying a telescope before buying a PC.*

We can select the episodes satisfying these constraints in post-processing steps after or during non-constraint episode mining. This approach, however, requires a very long execution time since mining algorithms often outputs a large number of frequent episodes. It is a better idea to use the constraint-based algorithm PPS introduced by Ma *et al.* [5]. PPS efficiently handles a *prefix anti-monotone constraint* [11], which means that if an episode satisfies the constraint, so does every prefix of the episode. Any shape constraint is prefix anti-monotonic, and PPS can extract partite episodes satisfying a given shape constraint in polynomial amortized time per output.

On the other hand, pattern constraints are not always prefix anti-monotonic. For example, a pattern constraint having *buying a telescope* in the first set and *buying a PC* in the third set, which is described above, is not prefix anti-monotonic one. This pattern constraint is not prefix anti-monotonic because there is an episode whose prefix does not satisfy it; It is obvious by considering the episode; *buying a telescope before buying a camera and then buying a PC* and it's prefix episodes.

If we allow an algorithm to add event to any part of the partite episode instead of adding only the tail of the episode, there is a possibility to enumerate episode satisfying any pattern constraint without generating candidate episodes. However, the efficient computation methods of episodes and their occurrence in that enumeration is non-trivial.

Main Results. Our goal is to present an algorithm that extracts frequent partite episodes satisfying a partwise constraint without post-processing steps. Then we show that our algorithm runs in $O(Nsc)$ time per partite episode and $O(Nsm)$ space, where N is the total size of an input sequence, s is the number

of event types in the input sequence (alphabet size), c is a constant that depends only on a given partwise constraint, and m is the maximum size of episodes.

Finally, our experimental result shows that our proposed algorithm is 230 times faster than the straightforward algorithm which consists of an algorithm for non-constraint episodes and post-processing steps for partwise constraints.

Related Works. In addition to Ma's research, there have been many studies on episode mining. Méger and Rigotti [7] introduced a different type of constraint called *gapmax* that represents the maximum time gap allowed between two events. Tatti *et al.* [13] and Zhou *et al.* [15] introduced *closed episode mining*. Closed episode mining is another approach to reduce the number of outputs and improve the mining efficiency. Closed episode mining algorithms extract only representative patterns called *closed episodes*, whereas constraint-based algorithms such as our algorithm and Méger's algorithm extract only episodes that satisfy some given conditions on which we focus. Seipel *et al.* [12] applied episode mining to system event logs for network management. Since their class of episodes was a subclass of partite episodes, their analysis would be improved by applying the proposed algorithm in this paper.

2 Partite Episodes

In this section, we introduce partite episodes and the related notions and lemmas necessary for a later discussion. We denote the sets of all natural numbers by $\mathbf{N}$. Then we define ∞ as the special largest number such that $a < \infty$ for all $a \in \mathbf{N}$. For a set $S = \{s_1, \ldots, s_n\}$ ($n \in \mathbf{N}$), we denote the cardinality n of S by $|S|$. For sets S and T, we denote the difference set $\{\, s \in S \mid s \notin T \,\}$ by $S \setminus T$. For a sequence $X = \langle x_1, \ldots, x_n \rangle$ ($n \in \mathbf{N}$), we denote the length n of X by $|X|$, the i-th element x_i of X by $X[i]$, and the consecutive subsequence $\langle x_i, \ldots, x_j \rangle$ of X by $X[i, j]$, for every $1 \leq i \leq j \leq n$, where we define $X[i, j] = \langle\rangle$ when $i > j$.

2.1 Input Event Sequence

Let $\Sigma = \{1, \ldots, m\}$ ($m \geq 1$) be a finite alphabet with the total order $\leq$ over $\mathbf{N}$. Each element $e \in \Sigma$ is called an *event*[1]. An *input event sequence* (an *input sequence*, for short) $\mathcal{S}$ on Σ is a finite sequence $\langle S_1, \ldots, S_\ell \rangle \in (2^\Sigma)^*$ of sets of events of length ℓ. For an input sequence $\mathcal{S}$, we define the *total size* $||\mathcal{S}||$ of $\mathcal{S}$ by $\sum_{i=1}^{\ell} |S_i|$. Clearly, $||\mathcal{S}|| = O(|\Sigma|\ell)$. Without loss of generality, we can assume that every event in Σ appears at least once in $\mathcal{S}$.

[1] Mannila *et al.* [6] originally referred to each element $e \in \Sigma$ itself as an *event type* and an occurrence of e as an *event*. However, we simply refer to both of these as *events*.

2.2 Partite Episodes

Mannila *et al.* [6] and Katoh *et al.* [3] formulated an episode as a partially ordered set and a labeled acyclic digraph, respectively. In this paper, for a sub-class of episodes called *partite episodes*, we define an episode as a sequence of sets of events for simpler representations.

Definition 1. A *partite episode* P over Σ is a sequence $\langle P_1, \ldots, P_k \rangle \in (2^{\Sigma})^*$ of sets of events ($k \geq 0$), where $P_i \subseteq \Sigma$ is called the i-th *part* for every $1 \leq i \leq k$. For a partite episode $P = \langle P_1, \ldots, P_k \rangle$, we define the *total size* $||P||$ of P by $\sum_{i=1}^{k} |P_i|$. We call P *proper* when $P_i \neq \emptyset$ for every $1 \leq i \leq k$.

Definition 2. *Let* $P = \langle \{a_{(1,1)}, \ldots, a_{(1,m_1)}\}, \ldots, \{a_{(k,1)}, \ldots, a_{(k,m_k)}\} \rangle$ *be a partite episode of length* $k \geq 0$*, and* $Q = \langle \{b_{(1,1)}, \ldots, b_{(1,n_1)}\}, \ldots, \{b_{(l,1)}, \ldots, a_{(l,n_l)}\} \rangle$ *a partite episode of length* $l \geq 0$*, where* $m_i \geq 0$ *for every* $1 \leq i \leq k$ *and* $n_i \geq 0$ *for every* $1 \leq i \leq l$*. A partite episode* P *is a* sub-episode *of* Q*, denoted by* $P \sqsubseteq Q$*, if and only if there exists some injective mapping* $h : A \to B$ *satisfying* (i) $A = \bigcup_{i=1}^{k} \bigcup_{j=1}^{m_i} \{(i,j)\}$, $B = \bigcup_{i=1}^{l} \bigcup_{j=1}^{n_i} \{(i,j)\}$, (ii) *for every* $x \in A$, $a_x = b_{h(x)}$ *holds, and* (iii) *for every* $(p_1, q_1), (p_2, q_2) \in A$*, and values* $(p'_1, q'_1) = h((p_1, q_1))$ *and* $(p'_2, q'_2) = h((p_2, q_2))$*, if* $p_1 < p_2$ *holds, then* $p'_1 < p'_2$ *holds.*

Let $P = \langle P_1, \ldots, P_k \rangle$ and $Q = \langle Q_1, \ldots, Q_l \rangle$ be partite episodes of length $k \geq 0$ and $l \geq 0$, respectively. We denote the partite episode $\langle P_1, \ldots, P_k, Q_1, \ldots, Q_l \rangle$ by $P \mapsto Q$, the partite episode $\langle P_1 \cup Q_1, \ldots, P_{\max(k,l)} \cup Q_{\max(k,l)} \rangle$ by $P \circ Q$, and the partite episode $\langle P_1 \setminus Q_1, \ldots, P_k \setminus Q_k \rangle$ by $P \setminus Q$, where we assume that $P_i = \emptyset$ for any $i > k$ and $Q_j = \emptyset$ for any $j > l$. In Fig. 1, we show examples of an input event sequence S and partite episodes P^i ($1 \leq i \leq 7$).

2.3 Occurrences

Next, we introduce occurrences of episodes in an input sequence. An *interval* in an input sequence $\mathcal{S}$ is a pair of integer $(s,t) \in \mathbf{N}^2$ satisfying $s \leq t$. Let P be a partite episode, and $x = (s,t)$ be an interval in an input sequence $\mathcal{S} = \langle S_1, \ldots, S_\ell \rangle$ ($\ell \geq 0$). A partite episode P *occurs in* an interval x, if and only if $P \sqsubseteq \mathcal{S}[s, t-1]$, where we define $\mathcal{S}[i]$ for any index such that $i < 1$ and $i > |\mathcal{S}|$ as $\mathcal{S}[i] = \emptyset$.

For a partite episode P that occurs in an interval $x = (s,t)$ in an input sequence $\mathcal{S}$, the interval x is a *minimum occurrence* if and only if P does not occur in both intervals $(s+1, t)$ and $(s, t-1)$. Moreover, a *minimum occurrence set* of P on $\mathcal{S}$ is a set $\{ (s,t) \mid P \sqsubseteq S[s, t-1], P \not\sqsubseteq S[s, t-2], P \not\sqsubseteq S[s+1, t-1] \} \subseteq \mathbf{N}^2$ of all minimum occurrences of P on $\mathcal{S}$. For a minimum occurrence set X of P, a *minimum occurrence list* (*mo-list*, for short) of P, denoted by $mo(P)$, is a sequence $\langle (s_1, t_1), \ldots, (s_n, t_n) \rangle \in (\mathbf{N}^2)^*$ of minimum occurrences such that $n = |X|$, $s_i < s_{i+1}$ for every $1 \leq i < n$.

A *window width* is a fixed positive integer $w \geq 1$. For an input sequence $\mathcal{S}$ and any $1 - w < i < |\mathcal{S}|$, the interval $(i, i+w)$ is a *window* of width w on $\mathcal{S}$. Then the frequency $freq_{\mathcal{S},w}(P)$ of a partite episode P is defined by the number of

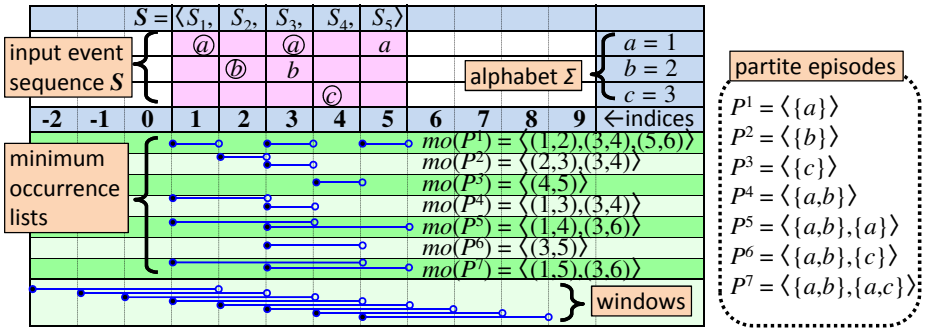


Fig. 1. An input event sequence $\mathcal{S}$ of length 5 (left top), partite episodes P^i $(1 \leq i \leq 7)$ (right), their minimum occurrence lists (left middle), and windows of width 4 on $\mathcal{S}$ (left bottom). In the input sequence $\mathcal{S}$, we indicate an occurrence of P^7 in the fourth window $(1, 5)$ in circles. See Example 1 for details.

windows (of width w on $\mathcal{S}$) in which P occurs. A *minimum frequency threshold* is any positive integer $\sigma \geq 1$. An episode P is *σ-frequent in $\mathcal{S}$* if $freq_{\mathcal{S},w}(P) \geq \sigma$. For a minimum frequency threshold σ and a window width w, the *minimum support threshold* $\acute{\sigma}$ is the relative value $\acute{\sigma} = \sigma/(|\mathcal{S}| + w - 1)$. By the definition of the frequency, we can show the next lemma.

Lemma 1. *Let P and Q be partite episodes, $\mathcal{S}$ an input sequence, and w a window width. If $P \sqsubseteq Q$ holds, then $freq_{\mathcal{S},w}(P) \geq freq_{\mathcal{S},w}(Q)$ holds.*

Our definition of frequency is identical to the one given by Mannila [6]. With this definition, an occurrence of episode could be counted more than once, and more than two occurrences could be counted only once. Lemma 1 holds even with another definition that does not have the problems above, although we do not explain why because of the limited space.

Example 1. In Fig. 1, we show an input sequence $\mathcal{S} = \langle\{a\}, \{b\}, \{a, b\}, \{c\}, \{a\}\rangle$ of length $\ell = 5$ over an alphabet $\Sigma = \{a, b, c\}$ of events, partite episodes $P^1 = \langle\{a\}\rangle$, $P^2 = \langle\{b\}\rangle$, $P^3 = \langle\{c\}\rangle$, $P^4 = P^1 \circ P^2 = \langle\{a, b\}\rangle$, $P^5 = P^4 \mapsto P^1 = \langle\{a, b\}, \{a\}\rangle$, $P^6 = P^4 \mapsto P^3 = \langle\{a, b\}, \{c\}\rangle$, and $P^7 = P^5 \circ P^6 = \langle\{a, b\}, \{a, c\}\rangle$, and their minimum occurrence lists $mo(P^1) = \langle(1, 2), (3, 4), (5, 6)\rangle$, $mo(P^2) = \langle(2, 3), (3, 4)\rangle$, $mo(P^3) = \langle(4, 5)\rangle$, $mo(P^4) = \langle(1, 3), (3, 4)\rangle$, $mo(P^5) = \langle(1, 4), (3, 6)\rangle$, $mo(P^6) = \langle(3, 5)\rangle$, and $mo(P^7) = \langle(1, 5), (3, 6)\rangle$, respectively, where $a = 1$, $b = 2$, and $c = 3$ are events. The input sequence $\mathcal{S}$ has eight windows with width 4 from $(-2, 2)$ to $(5, 9)$. Among these, the partite episode P^5 occurs in $(0, 4)$, $(1, 5)$, $(2, 6)$, and $(3, 7)$. Therefore, the frequency of P^5 is 4. Furthermore, the partite episode P^7 occurs in $(1, 5)$, $(2, 6)$, and $(3, 7)$. Therefore, the frequency of P^7 is 3. We see that $P^5 \sqsubseteq P^7$ and $freq_{\mathcal{S},4}(P^5) \geq freq_{\mathcal{S},4}(P^7)$.

3 Partwise Constraints

In this section, we introduce *partwise constraints* for partite episodes which consist of the *shape and pattern constraints*. A *shape constraint* for a partite episode P is a sequence $C = \langle c_1, \ldots, c_k \rangle \in \mathbf{N}^*$ $(k \geq 0)$ of natural numbers. A *pattern constraint* for a partite episode P is a sequence $D = \langle D_1, \ldots, D_k \rangle \in (2^{\Sigma})^*$ $(k \geq 0)$ of sets of events. We consider a pattern constraint as a partite episode.

Definition 3. A partite episode P *satisfies* the shape constraint C if $|P| \leq |C|$ and $|P[i]| \leq C[i]$ for every $1 \leq i \leq |C|$. A partite episode P *satisfies* the pattern constraint D if $|P| \geq |D|$ and $P[i] \supseteq D[i]$ for every $1 \leq i \leq |D|$.

By this definition, we see that a shape constraint is *anti-monotone* [10] and a pattern constraint is *monotone* [10].

Lemma 2. *Let P and Q be partite episodes such that $P \sqsubseteq Q$. For a shape constraint C and a pattern constraint D,* (i) *if Q is satisfying C then P is so, and* (ii) *if P is satisfying D then Q is so.*

We denote the classes of partite episodes (over Σ) by $\mathcal{PE}$. Moreover, we denote the classes of partite episodes satisfying a pattern constraint D by $\mathcal{PE}(D)$.

Definition 4. Partite Episode Mining with a Partwise Constraint: Given an input sequence $\mathcal{S} \in (2^{\Sigma})^*$, a window width $w \geq 1$, a minimum frequency threshold $\sigma \geq 1$, a shape constraint C, and a pattern constraint D, the task is to find all of the σ-frequent partite episodes satisfying C and D that occur in $\mathcal{S}$ with a window width w without duplication.

Our goal is to design an algorithm for the frequent partite episode mining problem with a partwise constraint, which we will show in the next section, in the framework of enumeration algorithms [2]. Let N be the total input size and M the number of solutions. An enumeration algorithm $\mathcal{A}$ is of *output-polynomial time*, if $\mathcal{A}$ finds all solutions in total polynomial time both in N and M. Moreover, $\mathcal{A}$ is of *polynomial delay*, if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a polynomial in N alone.

4 Algorithm

In this section, we present an output-polynomial time and a polynomial-space algorithm PartiteCD for extracting all the frequent partite episodes satisfying partwise constraints in an input event sequence. Throughout this section, let $\mathcal{S} = \langle S_1, \ldots, S_\ell \rangle \in (2^{\Sigma})^*$ be an input event sequence over an alphabet Σ, $w \geq 1$ a window width, and $\sigma \geq 1$ the minimum frequency threshold. Furthermore, let M be the number of all solutions of our algorithm, and m the maximum size of output episode P, that is, $m = \|P\| + |P|$. Then we define the length of the input sequence $\ell = |\mathcal{S}|$, the total size of the input sequence $N = ||\mathcal{S}|| + \ell$, the total size of the constraints $c = |C| + \|D\| + |D|$, and the alphabet size $s = |\Sigma|$ for analysis of time and space complexity of our algorithm, where we assume $O(s) = O(N)$ and $O(m) = O(N)$.

4.1 Family Tree

The main idea of our algorithm is to enumerate all of the frequent partite episodes satisfying partwise constraints by searching the whole search space from general to specific by using depth-first search. First, we define the search space. For a partite episode P such that $|P| \geq 1$, the *tail pattern* $tail(P)$ of P is defined by the partite episode Q such that $|Q| = \max\{\, 1 \leq i \leq |P| \,|\, P[i] \neq \emptyset \,\}$, $Q[i] = \emptyset$ for every $1 \leq i < |Q|$, and $Q[i] = \{\max P[i]\}$ for $i = |Q|$, where $\max P[i]$ is the maximum integer in the i-th part of P. Then, for a pattern constraint D, we introduce the parent-child relationship between partite episodes satisfying D.

Definition 5. The partite episode $\bot = D$ is the *root*. The *parent* of the partite episode $P = \langle P_1, \ldots, P_k \rangle$ is defined by:

$$parent_D(P) = \begin{cases} \langle P_1, \ldots, P_{k-1} \rangle, & \text{if } |P_k| \leq 1 \text{ and } |P| > |D|, \\ P \setminus tail(P \setminus D), & \text{otherwise.} \end{cases}$$

We define the set of all children of P by $children_D(P) = \{\, Q \,|\, parent_D(Q) = P \,\}$. Then we define the *family tree* for $\mathcal{PE}(D)$ by a rooted digraph $\mathcal{T}(\mathcal{PE}(D)) = (V, E, \bot)$ with the root $\bot$, where the root $\bot = D$, the vertex set $V = \mathcal{PE}(D)$, and the edge set $E = \{\, (P, Q) \,|\, P = parent_D(Q), Q \neq D \,\}$.

Lemma 3. *The family tree $\mathcal{T}(\mathcal{PE}(D)) = (V, E, D)$ for $\mathcal{PE}(D)$ is a rooted tree with the root D.*

For any episode Q on $\mathcal{T}(\mathcal{PE}(D))$, the parent $parent_D(Q)$ is a subepisode of Q. Therefore, we can show the next lemma by Lemma 1 and Lemma 2.

Lemma 4. *Let C be a shape constraint, D a pattern constraint, and P and Q partite episodes such that $P = parent_D(Q)$. If Q is frequent then so is P. If Q is satisfying C then so is P.*

By Lemma 2 and Lemma 3, we know that a family tree $\mathcal{T}(\mathcal{PE}(D))$ contains only episodes satisfying a pattern constraint D. Thus, we can make a search tree containing only episodes that are frequent and satisfy the shape and pattern constraints by pruning episodes that do not satisfy the conditions.

Example 2. We describe the part of family trees $\mathcal{T}(\mathcal{PE}(\langle\rangle))$ and $\mathcal{T}(\mathcal{PE}(\langle\{\}, \{b\}\rangle))$ that forms the spanning trees for all partite episodes of $\mathcal{PE}(\langle\rangle)$ and $\mathcal{PE}(\langle\{\}, \{b\}\rangle)$ on an alphabet $\Sigma = \{a, b\}$ in Fig. 2. For a pattern constraint $D = \langle\{\}, \{b\}\rangle$, the parent of $P^1 = \langle\{a, b\}, \{a, b\}\rangle$ is $P^1 \setminus tail(P^1 \setminus D) = P^1 \setminus tail(\langle\{a, b\}, \{a\}\rangle) = P^1 \setminus \langle\{\}, \{a\}\rangle = \langle\{a, b\}, \{b\}\rangle = P^2$.

4.2 Pattern Expansion and Incremental Computation

Secondly, we discuss how to enumerate all children of a parent. For a pattern constraint D, a partite episode Q, and its parent P, we define the *index of the expanded part* of Q by $iex(Q) = |Q|$ if $|Q| > |P|$, and $iex(Q) = |tail(Q \setminus P)|$ otherwise. Additionally, we define $iex(Q) = 0$ for the root $Q = D$.

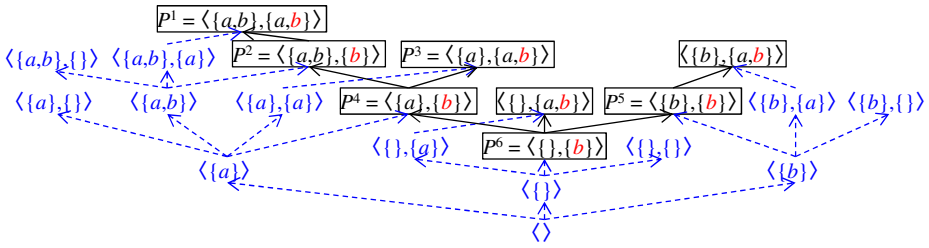


Fig. 2. The parent-child relationships on an alphabet $\Sigma = \{a, b\}$ for pattern constraints $\langle\rangle$ (dashed arrows) and $\langle\{\}, \{b\}\rangle$ (solid arrows), where $a = 1$ and $b = 2$ are events.

Let $h = iex(P)$ be the index of the expanded part of a parent episode P. We define *type-i children* of P by $children_D(P, i) = \{\, Q \in children_D(P) \mid iex(Q) = i, \|Q\| > \|P\| \,\}$ for an index $i \geq h$. By Definition 5, we can make any type-i child Q of P by adding an event $a \in \Sigma \setminus (P[i] \cup D[i])$ at $P[i]$. Moreover, by Definition 5, we see $children_D(P) = (\bigcup_{i=m}^{n} children_D(P, i)) \cup \{P \mapsto \langle\emptyset\rangle\}$, where $m = \max(h, 1)$, and $n = \max(h, |D|) + 1$.

Furthermore, for partite episodes P, Q, and S such that $P = parent_D(Q)$, $S = parent_D(P)$, $iex(Q) \geq 1$, and $|(Q \setminus D)[iex(Q)]| \geq 2$, we define the *uncle* R of Q by $uncle_D(Q) = S \circ tail(Q \setminus D)$.

Example 3. In Fig. 2, for a pattern constraint $D = \langle\{\}, \{b\}\rangle$, an episode $P^4 = \langle\{a\}, \{b\}\rangle$ is the parent of an episode $P^2 = \langle\{a, b\}, \{b\}\rangle$. The index $iex(P^2)$ of the expanded part of P^2 is $|tail(P^2 \setminus P^4)| = |tail(\langle\{b\}, \{\}\rangle)| = |\langle\{b\}\rangle| = 1$. Therefore, P^2 is a type-1 child of P^4. On the other hand, P^3 is a type-2 child of P^4 because $iex(P^3)$ is $|tail(\langle\{\}, \{b\}\rangle)| = 2$. Since the parent of P^4 is P^6, the uncle of P^2 is $P^6 \circ tail(P^2 \setminus D) = P^6 \circ \langle\{a\}\rangle = \langle\{a\}, \{b\}\rangle = P^5$.

To compute a type-i child Q of P and its mo-list incrementally from the parent P, we make Q as follows. Let $h = iex(P)$ be the index of the expanded part of P; (i) we make Q by $P \circ R$, and the subepisode $Q_{sub} = Q[1, i]$ of Q by $P_{sub} \circ R_{sub}$ when $i = h$, where $R = uncle_D(Q)$, $P_{sub} = P[1, i]$, and $R_{sub} = R[1, i]$; (ii) we make Q by $P_{sub} \mapsto D_H \mapsto A \mapsto D_T$ and the subepisode $Q_{sub} = Q[1, i]$ of Q by $P_{sub} \mapsto D_H \mapsto A$ when $i > h$, where $P_{sub} = P[1, h]$, $D_H = D[h + 1, i - 1]$, $A = D[i] \circ \langle\{a\}\rangle$, and $D_T = D[i + 1, |D|]$. Since the length of mo-list I of an episode P is less than the length $|\mathcal{S}|$ of the input sequence $\mathcal{S}$, we can incrementally compute the mo-list in $O(\|\mathcal{S}\|) = O(N)$ time [3, 4, 8].

4.3 Depth-first Enumeration

In Fig. 3, we describe the algorithm PartiteCD, and its subprocedure RecCD for extracting frequent partite episodes satisfying a partwise constraint. For a pattern constraint D, the algorithm is a backtracking algorithm that traverses the spanning tree $\mathcal{T}(\mathcal{PE}(D))$ based on a depth-first search starting from the root $P = D$ using the parent-child relationships over $\mathcal{PE}(D)$.

algorithm PARTITECD($\mathcal{S}, \Sigma, w, \sigma, C, D$)
input: input sequence $\mathcal{S} \in (2^{\Sigma})^*$, alphabet of events Σ, window width $w > 0$, minimum frequency $1 \leq \sigma$, shape and pattern constraints C and D ,respectively;
output: frequent partite episodes satisfying the shape and pattern constraints;
01 *compute the mo-list* $mo(a)$ *for each event* a *in* Σ
by scanning imput sequence $\mathcal{S}$ *at once and store to* $\Sigma_{\mathcal{S}}$;
02 *make the root episode* $P = D$ *and compute its mo-list* $mo(P)$ *from* $\Sigma_{\mathcal{S}}$;
03 **if** (P *is frequent and satisfying* C) RECCD($P, 1, \emptyset, \Sigma_{\mathcal{S}}, w, \sigma, C, D$);

procedure RECCD($P, i, U, \Sigma_{\mathcal{S}}, w, \sigma, C, D$)
output: all frequent partite episodes that are descendants of P;
04 **output** P;
05 **while** ($i \leq \max(h, |D|) + 1$) **do** // where $h = iex(P)$.
06 $V := \emptyset$;
07 **foreach** ($a \in \Sigma \setminus (P[i] \cup D[i])$) **do**
08 *make a type-*i *child* Q *of* P *by adding the event* a *at* $P[i]$;
09 *compute the mo-lists of* Q *and the subepisode* $Q_{sub} = Q[1, i]$ *by using* U;
10 **if** (Q *is frequent and satisfying* C) *store* $(Q, mo(Q), mo(Q_{sub}))$ to V;
11 **end foreach**
12 **foreach** (a *child* Q *of* P *stored in* V) RECCD($Q, i, V, \Sigma_{\mathcal{S}}, w, \sigma, C, D$);
13 $i := i + 1$;
14 **end while**
15 **if** ($P \mapsto \langle\emptyset\rangle$ *is frequent and satisfying* C) RECCD($P \mapsto \langle\emptyset\rangle, i, \emptyset, \Sigma_{\mathcal{S}}, w, \sigma, C, D$);

Fig. 3. The main algorithm PARTITECD and a recursive subprocedure RECCD for mining frequent partite episodes satisfying shape and pattern constraints.

First, for an alphabet Σ, PARTITECD computes mo-lists $mo(\langle\{a\}\rangle)$ of partite episodes of size 1 for every event $a \in \Sigma$. Then PARTITECD makes the root episode $P = D$ and calls the recursive subprocedure RECCD. Finally, for an episode P the recursive subprocedure RECCD enumerates all frequent episodes that are descendants of P and satisfy the shape constraint C. Algorithm PARTITECD finds all of the frequent partite episodes satisfying the partwise constraints occurring in an input event sequence until all recursive calls are finished.

Theorem 1. *Algorithm* PARTITECD *runs in* $O(Nsc)$ *amortized time per output and in* $O(Nsm)$ *space.*

Finally, we describe the possible improvement for PARTITECD. By using the alternating output technique [14], that is, we output episodes after Line 15 (the end of RECCD) instead of Line 4 if and only if the depth of recursions is even, our algorithm runs in $O(Nsc)$ delay.

5 Experimental Results

In this section, we show that the algorithm described in Sect. 4 has a significantly better performance than a straightforward algorithm which consists of an algorithm for non-constraint episodes and post-processing steps for partwise constraints by experiments for an artificial data set.

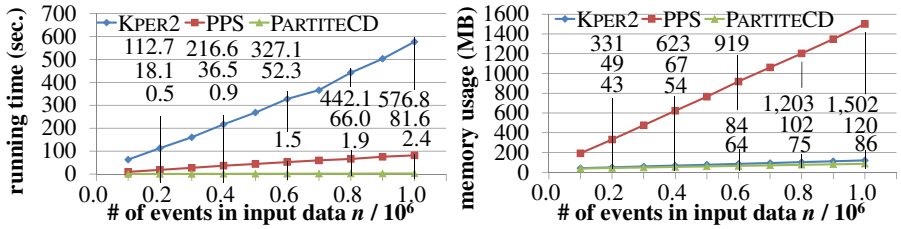


Fig. 4. Running time (left) and memory usage (right) for the number of events in input data.

5.1 Data and Method

An artificial data set consisted of the randomly generated event sequence $\mathcal{S} = \langle S_1, \ldots, S_\ell \rangle$ ($\ell \geq 1$) over an alphabet $\Sigma = \{1, \ldots, s\}$ ($s \geq 1$) as described below. Let $1 \leq i \leq \ell$ be any index. For every event $a \in \Sigma$, we add a into S_i independently with a probability of $p = 0.1/a$. By repeating this process for every S_i, we made a skewed data set that emulates real-world data sets having long tails such as point-of-sale data sets.

We implemented the following algorithms to compare their performance. KPER2 is a straightforward algorithm which consists of our previous algorithm [3] for non-constraint episodes and post-processing steps for partwise constraints. PPS is an algorithm which consists of Ma's algorithm [5] handling prefix anti-monotone constraints and post-processing steps for partwise constraints which are not prefix anti-monotonic. PPS efficiently handles shape constraints and a part of pattern constraints which have prefix anti-monotonic property, that is, for a partite episode P, a pattern constraint D, and an index $1 \leq i \leq |P|$, if $P[1, i]$ does not satisfy $D[1, i]$, then PPS does not generate P. PARTITECD is our algorithm that handles partwise constraints presented in Sect. 4.

All the experiments were run on a Xeon X5690 3.47GHz machine with 180 GB of RAM running Linux (CentOS 6.3). If not explicitly stated otherwise, we assume that the number of event in the input sequence S is $n = \|\mathcal{S}\| = 1,000,000$, the alphabet size is $s = |\Sigma| = 1,000$, the window width is $w = 20$, the minimum support threshold is $\acute{\sigma} = 0.1\%$, the shape constraint $C = \langle +\infty, 1, +\infty \rangle$, and the pattern constraint $D = \langle \{10\}, \emptyset, \{10\} \rangle$.

5.2 Experiments

Figure 4 shows the running time (left), and the amount of memory usage (right) of the algorithms KPER, PPS, and PARTITECD for the number of events n in input data. With respect to the total running time, we see that PARTITECD is 230 times as fast as KPER2 and 35 times as fast as PPS in this case. This difference comes from the number of episode generated by each algorithm before post-processing steps.

For the amount of memory usage, we see that PPS uses more memory than both KPAR2 and PARTITECD. The reason is that PPS is based on breadth-first

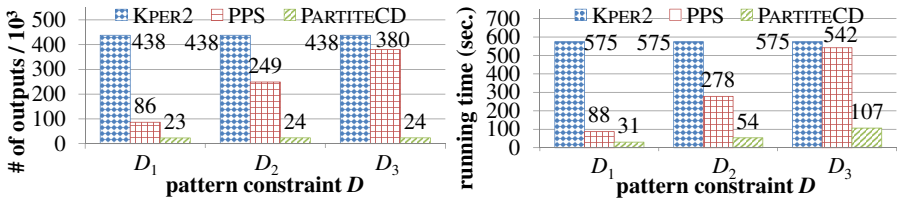


Fig. 5. The number of outputs before post-prossesing steps (left) and running time (rigth) for pattern constraints D_i $(1 \leq i \leq 3)$, where $D_1 = \langle\{10\}, \emptyset, \emptyset\rangle$, $D_2 = \langle\emptyset, \{10\}, \emptyset\rangle$, and $D_3 = \langle\emptyset, \emptyset, \{10\}\rangle$.

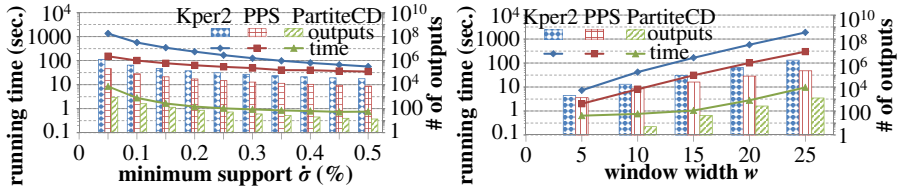


Fig. 6. Running time and number of outputs against minimum supports (left) and window width (right).

search in a search space [5], whereas both KPAR2 and PARTITECD are based on depth-first search. We also see that, both the running time and the memory usage of these algorithms seem to be almost linear to the input size and must therefore scale well on large datasets.

Figure 5 shows the number of outputs before the post-processing steps (left) and the total running time (right) for pattern constraints D_i $(1 \leq i \leq 3)$. In this experiment, we used a shape constraint $C = \langle+\infty, +\infty, +\infty\rangle$. We see that the number of outputs before post-processing steps of PPS is large and the running time is long if a part $\{10\}$ of the pattern constraint D appears at the tail side of D. The reason is that since PPS reduces the number of outputs by pruning based on prefix anti-monotonic property, PPS generates a large number of episodes before pruning by the constraint that appears at the tail side. On the other hand, the number of outputs of PARTITECD is almost constant for every constraint D, because PARTITECD generates only episodes that satisfy the given constraints.

Figure 6 shows the total running time and the number of outputs before the post-processing steps against minimum support $\dot{\sigma}$ (left) and window width w (right). We see that PARTITECD outperforms other algorithms both on the number of outputs and the running time for every parameter set.

Overall, we conclude that the partwise constraint reduces the number of outputs and the proposed algorithm handles the constraint much more efficiently than the straightforward algorithm using post-processing steps. In other words, by using our algorithm with our constraint, we can extract partite episodes on which we focused with lower frequency thresholds for larger input data in the same running time.

6 Conclusion

This paper studied the problem of frequent partite episode mining with partwise constraints. We presented an algorithm that finds all frequent partite episodes satisfying a partwise constraint in an input sequence. Then, we showed that our algorithm runs in the output polynomial time and polynomial space. We reported the experimental results that compare the performance of our algorithm and other straightforward algorithms. Both the theoretical and experimental results showed that our algorithm efficiently extracts episodes on which we focused. Thus, we conclude that our study improved the efficiency of data mining.

A possible future path of study will be the extension of PartiteCD for the class of proper partite episodes to reduce redundant outputs by traversing a search tree containing only proper partite episodes satisfying a pattern constraint. Our future work will also include the application of our algorithm to real-world data sets.

References

1. Arimura, H., Uno, T.: A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. In: ISAAC, LNCS 3827. pp. 724–737. Springer-Verlag (2005)
2. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65, 21–46 (1996)
3. Katoh, T., Arimura, H., Hirata, K.: Mining frequent k-partite episodes from event sequences. In: LLLL, LNAI 6284. pp. 331–344. Springer-Verlag (2010)
4. Katoh, T., Hirata, K.: A simple characterization on serially constructible episodes. In: PAKDD, LNAI 5012. pp. 600–607. Springer-Verlag (2008)
5. Ma, X., Pang, H., Tan, K.L.: Finding constrained frequent episodes using minimal occurrences. In: ICDM. pp. 471–474 (2004)
6. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery 1(3), 259–289 (1997)
7. Méger, N., Rigotti, C.: Constraint-based mining of episode rules and optimal window sizes. In: PKDD. pp. 313–324 (2004)
8. Ohtani, H., Kida, T., Uno, T., Arimura, H.: Efficient serial episode mining with minimal occurrences. In: ICUIMC. pp. 457–464 (2009)
9. Pei, J., Han, J., Mortazavi-Asi, B., Wang, J.: Mining sequential patterns by pattern-growth: The PrefixSpan approach. IEEE Transactions on Knowledge and Data Engineering 16(11), 1–17 (2004)
10. Pei, J., Han, J.: Can we push more constraints into frequent pattern mining? In: KDD. pp. 350–354 (2000)
11. Pei, J., Han, J., Wang, W.: Mining sequential patterns with constraints in large databases. In: CIKM. pp. 18–25. ACM (2002)
12. Seipel, D., Köhler, S., Neubeck, P., Atzmueller, M.: Mining complex event patterns in computer networks. In: NFMCP, LNAI 7765. Springer Verlag (2012)
13. Tatti, N., Cule, B.: Mining closed strict episodes. Data Mining and Knowledge Discovery 25(1), 34–66 (2012)
14. Uno, T.: Two general methods to reduce delay and change of enumeration algorithms. Tech. rep., National Institute of Informatics (2003)
15. Zhou, W., Liu, H., Cheng, H.: Mining closed episodes from event sequences efficiently. In: PAKDD. pp. 310–318 (2010)

Conditional Log-Likelihood for Continuous Time Bayesian Network Classifiers

Daniele Codecasa and Fabio Stella

DISCo, Università degli Studi di Milano-Bicocca,
Viale Sarca 336, 20126 Milano, Italy
{codecasa,stella}@disco.unimib.it

Abstract. Continuous time Bayesian network classifiers are designed for analyzing multivariate streaming data when time duration of events matters. New continuous time Bayesian network classifiers are introduced while their conditional log-likelihood scoring function is developed. A learning algorithm, combining conditional log-likelihood with Bayesian parameter estimation is developed. Classification accuracy values achieved on synthetic and real data by continuous time and dynamic Bayesian network classifiers are compared. Numerical experiments show that the proposed approach outperforms dynamic Bayesian network classifiers and continuous time Bayesian network classifiers learned with log-likelihood.

Keywords: Multivariate streaming data, conditional log-likelihood.

1 Introduction

Streaming data are relevant to *finance*, with reference to high frequency trading [4], *computer science*, with reference to system error logs, web search query logs, network intrusion detection, social networks [23] and temporal semantic [15], and *engineering*, with reference to image, audio and video processing [30]. They are also important for analyzing GPS data as shown in [14] and [3] where buses and animals paths are analyzed. Streaming data are becoming increasingly important in medicine for patient monitoring and continuous time diagnosis including the study of firing pattern of neurons [27]. Finally, they are becoming relevant in biology where time course data [1] allow the reconstruction of gene regulatory networks, to model the evolution of infections, and to learn and analyze metabolic networks [28].

Dynamic Bayesian networks (DBNs) [5] and hidden Markov models (HMMs) [19] offer a natural way to represent and analyze streaming data. However, DBNs are concerned with discrete time and thus suffer from several limitations due to the fact that it is not clear how timestamps should be discretized. In the case where a too slow sampling rate is used the data will be poorly represented while a too fast sampling rate rapidly makes learning and inference prohibitive. Furthermore, it has been pointed out [12] that when allowing long term dependencies it is required to condition on multiple steps into the past, and thus choosing a too fast sampling rate increases the number of such steps that need to be conditioned on.

Continuous time Bayesian networks (CTBNs) [16], continuous time noisy-or (CT-NOR) [22], Poisson cascades [23] and Poisson networks [20] together with the piecewise-constant conditional intensity model (PCIM) [12] are interesting models to represent and analyze continuous time processes. CT-NOR and Poisson cascades are devoted to model event streams while they require the modeler to specify a parametric form for temporal dependencies. This aspect significantly impacts performance and the problem of model selection in CT-NOR and Poisson cascades has not been addressed yet. This limitation is overcame by PCIMs which perform structure learning to model how events in the past affect future events of interest. CTBNs are homogeneous Markov models which allow to represent joint trajectories of discrete finite variables.

In this paper we consider the problem of *temporal classification*, where data stream measurements are available over a period of time in history while the class is expected to occur in the future. This kind of problem can be addressed by discrete and continuous time models. Discrete time models include dynamic latent classification models [31], a specialization of the latent classification model (LCM) [13], and DBNs [5]. Continuous time models, as continuous time Bayesian network classifiers (CTBNCs) [24], overcame the problem of timestamps discretization. The main contributions of the paper are:

- definition of new classifiers from the class of CTBNCs,
- development of the conditional log-likelihood scoring function for CTBNCs,
- performance comparison of CTBNCs learned with the conditional log-likelihood score to CTBNCs learned with log-likelihood score and to DBN classifiers.

The paper is organized as follows; Section 2 is devoted to notations and definitions. New classifiers are introduced and analyzed in Section 3. Section 4 concerns numerical experiments where synthetic datasets generated from models of increasing complexity are used. In this section a real dataset on post-stroke rehabilitation is analyzed. Finally, Section 5 is devoted to comments.

2 Continuous time classification

2.1 Continuous Time Bayesian Networks

Dynamic Bayesian networks (DBNs) model dynamic systems without representing time explicitly. They discretize time to represent a dynamic system through several time slices. In [17] the authors pointed out that "*since DBNs slice time into fixed increments, one must always propagate the joint distribution over the variables at the same rate*" . Therefore, if the system consists of processes which evolve at different time granularities and/or the obtained observations are irregularly spaced in time, the inference process may become computationally intractable.

Continuous time Bayesian networks (CTBNs) overcome the limitations of DBNs by explicitly representing temporal dynamics and thus allow us to recover the probability distribution over time when specific events occur. CTBNs have been used to discover intrusion in computers [29], to analyze the reliability of

dynamic systems [2], for learning social networks dynamics [8] and to model cardiogenic heart failure [10]. A continuous time Bayesian network (CTBN) is a probabilistic graphical model whose nodes are associated with random variables and whose state evolves continuously over time.

Definition 1. *(Continuous time Bayesian network). [17]. Let* $\mathbf{X}$ *be a set of random variables* $X_1, X_2, ..., X_N$*. Each* X_n *has a finite domain of values* $Val(X_n) = \{x_1, x_2, ..., x_I\}$*. A continuous time Bayesian network* $\aleph$ *over* $\mathbf{X}$ *consists of two components: the first is an initial distribution* $P^0_{\mathbf{X}}$*, specified as a Bayesian network* $\mathcal{B}$ *over* $\mathbf{X}$*. The second is a continuous transition model, specified as:*

- *a directed (possibly cyclic) graph* $\mathcal{G}$ *whose nodes are* $X_1, X_2, ..., X_N$*;* $Pa(X_n)$ *denotes the parents of* X_n *in* $\mathcal{G}$*.*
- *a conditional intensity matrix,* $\mathbf{Q}_{X_n}^{Pa(X_n)}$*, for each variable* $X_n \in \mathbf{X}$*.*

Given the random variable X_n, the *conditional intensity matrix* (CIM) $\mathbf{Q}_{X_n}^{Pa(X_n)}$ consists of a set of intensity matrices, one intensity matrix

$$\mathbf{Q}_{X_n}^{pa(X_n)} = \begin{bmatrix} -q_{x_1}^{pa(X_n)} & q_{x_1x_2}^{pa(X_n)} & . & q_{x_1x_I}^{pa(X_n)} \\ q_{x_2x_1}^{pa(X_n)} & -q_{x_2}^{pa(X_n)} & . & q_{x_2x_I}^{pa(X_n)} \\ . & . & . & . \\ q_{x_Ix_1}^{pa(X_n)} & q_{x_Ix_2}^{pa(X_n)} & . & -q_{x_I}^{pa(X_n)} \end{bmatrix},$$

for each instantiation $pa(X_n)$ of the parents $Pa(X_n)$ of node X_n, where $q_{x_i}^{pa(X_n)} = \sum_{x_j \neq x_i} q_{x_ix_j}^{pa(X_n)}$ is the rate of leaving state x_i for a specific instantiation $pa(X_n)$ of $Pa(X_n)$, while $q_{x_ix_j}^{pa(X_n)}$ is the rate of arriving to state x_j from state x_i for a specific instantiation $pa(X_n)$ of $Pa(X_n)$. Matrix $\mathbf{Q}_{X_n}^{pa(X_n)}$ can equivalently be summarized by using two types of parameters, $q_{x_i}^{pa(X_n)}$ which is associated with each state x_i of the variable X_n when its parents are set to $pa(X_n)$, and $\theta_{x_ix_j}^{pa(X_n)} = \frac{q_{x_ix_j}^{pa(X_n)}}{q_{x_i}^{pa(X_n)}}$ which represents the probability of transitioning from state x_i to state x_j, when it is known that the transition occurs at a given instant in time.

Example 1. Figure 1 shows a part of the drug network introduced in [17]. It contains a cycle, indicating that whether a person is hungry (H) depends on how full his/her stomach (S) is, which depends on whether or not he/she is eating (E), which in turn depends on whether he/she is hungry. We assume that E and H are binary variables (i.e. no (n)/yes (y)) while S is ternary (i.e. full (f)/average (a)/empty (e)). Then, the CIMs for E are the [2x2] matrices $\mathbf{Q}_E^n$, and $\mathbf{Q}_E^y$, the CIMs for S are the [3x3] matrices $\mathbf{Q}_S^n$ and $\mathbf{Q}_S^y$, while the CIMs for H are the [2x2] matrices $\mathbf{Q}_H^f$, $\mathbf{Q}_H^a$ and, $\mathbf{Q}_H^e$. For matters of brevity we only show $\mathbf{Q}_S^y$ with two equivalent parametric representations:

$$\mathbf{Q}_S^y = \begin{bmatrix} -q_f^y & q_{fa}^y & q_{fe}^y \\ q_{af}^y & -q_a^y & q_{ae}^y \\ q_{ef}^y & q_{ea}^y & -q_e^y \end{bmatrix} = \begin{bmatrix} -.03 & .02 & .01 \\ 5.99 & -6 & .01 \\ 1 & 5 & -6 \end{bmatrix} \quad (1)$$

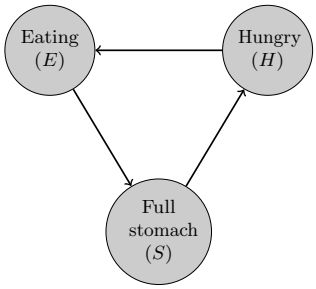


Fig. 1. A part of the drug network.

$$\mathbf{Q}_I^y = \begin{bmatrix} q_f^y & 0 & 0 \\ 0 & q_a^y & 0 \\ 0 & 0 & q_e^y \end{bmatrix} \left(\begin{bmatrix} 0 & \theta_{fa}^y & \theta_{fe}^y \\ \theta_{af}^y & 0 & \theta_{ae}^y \\ \theta_{ef}^y & \theta_{ea}^y & 0 \end{bmatrix} - \mathbf{I} \right) = \begin{bmatrix} .03 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix} \left(\begin{bmatrix} 0 & \frac{.02}{.03} & \frac{.01}{.03} \\ \frac{5.99}{6} & 0 & \frac{.01}{6} \\ \frac{1}{6} & \frac{5}{6} & 0 \end{bmatrix} - \mathbf{I} \right) \quad (2)$$

where $\mathbf{I}$ is the identity matrix. If we view units of time as hours, then we expect a person who has an empty stomach (S=e) and is eating (E=y) to stop having an empty stomach in 10 minutes ($\frac{1}{6}$ hour). The stomach will then transition from state e (S=e) to state a (S=a) with probability $\frac{5}{6}$ and to state f (S=f) with probability $\frac{1}{6}$. Equation 1 is a compact representation of the CIM while Equation 2 is useful because it explicitly represents the transition probability value from state x to state x', i.e. $\theta_{xx'}^{pa(X)}$.

CTBNs allow two types of evidence, namely *point evidence* and *continuous evidence*, while HMMs and DBNs allow only point evidence. *Continuous evidence* is the knowledge of the states of a set of variables $\mathbf{X}$ throughout an entire half-closed interval of time $[t_1, t_2)$: $\mathbf{Z}^{[t_1,t_2)} = \mathbf{z}^{[t_1,t_2)}$, where $\mathbf{Z}^{[t_1,t_2)} = (X_1^{[t_1,t_2)}, X_2^{[t_1,t_2)}, ..., X_k^{[t_1,t_2)})$ while $\mathbf{z}^{[t_1,t_2)} = (x_1^{[t_1,t_2)}, x_2^{[t_1,t_2)}, ..., x_k^{[t_1,t_2)})$.

Inference in CTBNs can be performed by exact and approximate algorithms. *Full amalgamation* [17] allows exact inference by generating an exponentially-large matrix representing the transition model over the entire state space. Exact inference in CTBNs is known to be NP-hard, and thus different approximate algorithms have been proposed. In [16] the authors introduced the *Expectation Propagation* algorithm (EP), while in [21] an optimized variant of EP is presented. Alternatives are offered by sampling based inference algorithms such as *importance sampling* algorithm [7] and *Gibbs sampling* algorithm [6].

Given the dataset $\mathcal{D}$, parameter learning is similar to *maximum likelihood estimation*, but accounts for the *imaginary counts* $\alpha_x^{pa(X)}$, $\alpha_{xx'}^{pa(X)}$ and, $\tau_x^{pa(X)}$ of the hyperparameters:

$$q_x^{pa(X)} = \frac{\alpha_x^{pa(X)} + M[x \mid pa(X)]}{\tau_x^{pa(X)} + T[x \mid pa(X)]}; \; \theta_{xx'}^{pa(X)} = \frac{\alpha_{xx'}^{pa(X)} + M[x, x' \mid pa(X)]}{\alpha_x^{pa(X)} + M[x \mid pa(X)]} \quad (3)$$

where $M[x, x' \mid pa(X)]$, $M[x \mid pa(X)]$ and $T[x \mid pa(X)]$ are the *sufficient statistics*. $M[x, x' \mid pa(X)]$ is the count of transitions from state x to state x' for

node X when the state of its parents $Pa(X)$ is set to $pa(X)$. $M[x \mid pa(X)] = \sum_{x' \neq x} M[x, x' \mid pa(X)]$ is the count of transitions leaving state x of node X when the state of its parents $Pa(X)$ is set to $pa(X)$. Finally, $T[x \mid pa(X)]$ represents the time spent in state x by the variable X when the state of its parents $Pa(X)$ is set to $pa(X)$.

Learning the structure of a CTBN from a given dataset $\mathcal{D}$ has been addressed as an optimization problem over possible CTBN structures [18]. It consists of finding the structure $\mathcal{G}$ which maximizes the following score:

$$\mathbf{score}_{\aleph}(\mathcal{G} : \mathcal{D}) = \ln P(\mathcal{D}|\mathcal{G}) + \ln P(\mathcal{G}). \tag{4}$$

However, the search space of this optimization problem is significantly simpler than that of BNs or DBNs. Indeed, it is known that learning the optimal structure of a BN is NP-hard, while the same does not hold true in the context of CTBNs where all edges are across time and thus represent the effect of the current value of one variable on the next value of the other variables. Therefore, no acyclicity constraints arise, and it is possible to optimize the parent set for each variable of the CTBN independently.

2.2 Continuous Time Bayesian Network Classifiers

Continuous time Bayesian network classifiers (CTBNCs) [24] are a specialization of CTBNs. They allow polynomial time classification inference which is NP-hard for general CTBNs. Classifiers from this class explicitly represent the evolution in continuous time of the set of random variables X_n, $n = 1, 2, ..., N$ which are assumed to depend on the class node Y.

Definition 2. *(Continuous time Bayesian network classifier)*[1]*. A continuous time Bayesian network classifier is a pair* $\mathcal{C} = \{\aleph, P(Y)\}$ *where* $\aleph$ *is a CTBN model with attribute nodes* $X_1, X_2, ..., X_N$*,* Y *is the class node with marginal probability* $P(Y)$ *on states* $Val(Y) = \{y_1, y_2, ..., y_K\}$*,* $\mathcal{G}$ *is the graph of the CTBNC, such that the following conditions hold:*

- $Pa(Y) = \emptyset$*, the class variable* Y *is associated with a root node;*
- Y *is fully specified by* $P(Y)$ *and does not depend on time.*

Given a dataset $\mathcal{D}$ with no missing data, a CTBNC is learned by maximizing the score (4) subjected to the constraints listed in Definition 2. However, exact learning requires to set in advance the maximum number of parents k for the nodes $X_1, X_2, ..., X_N$ [16]. Therefore, in the case where k is not small a considerable computational effort is required to find the graph structure $\mathcal{G}^*$ maximizing the score (4). In such a case we resort to hill-climbing or to the continuous time naive Bayes classifier.

[1] This definition differs from the one proposed in [24]. In fact, we do not require the CTBNC graph to be connected. Thus, features selection is obtained as by product of CTBNC structural learning.

Definition 3. *(Continuous time naive Bayes classifier). [24] A continuous time naive Bayes classifier is a continuous time Bayesian network classifier* $\mathcal{C} = \{\aleph, P(Y)\}$ *such that* $Pa(X_n) = \{Y\}$, $n = 1, 2, ..., N$.

According to [24] a CTBNC $\mathcal{C} = \{\aleph, P(Y)\}$ classifies a stream of continuous time evidence $\mathbf{z} = (x_1, x_2, ..., x_N)$ for the attributes $\mathbf{Z} = (X_1, X_2, ..., X_N)$ over J contiguous time intervals, i.e. a stream of continuous time evidence $\mathbf{Z}^{[t_1,t_2)} = \mathbf{z}^{[t_1,t_2)}$, $\mathbf{Z}^{[t_2,t_3)} = \mathbf{z}^{[t_2,t_3)}$, ..., $\mathbf{Z}^{[t_{J-1},t_J)} = \mathbf{z}^{[t_{J-1},t_J)}$, by selecting the value y^* for the class Y which maximizes the posterior probability $P(Y|\mathbf{z}^{[t_1,t_2)}, \mathbf{z}^{[t_2,t_3)}, ..., \mathbf{z}^{[t_{J-1},t_J)})$, which is proportional to

$$P(Y) \prod_{j=1}^{J} q^{pa(X_{m_j})}_{x^j_{m_j} x^{j+1}_{m_j}} \prod_{n=1}^{N} exp\left(-q^{pa(X_n)}_{x^j_n} \delta_j\right), \tag{5}$$

where:

- $\delta_j = t_j - t_{j-1}$ is the length of the j^{th} time interval of the stream $\mathbf{z}^{[t_1,t_2)}, \mathbf{z}^{[t_2,t_3)}$, ..., $\mathbf{z}^{[t_{J-1},t_J)}$ of continuous time evidence;
- $q^{pa(X_n)}_{x^j_n}$ is the parameter associated with state x^j_n, in which the variable X_n was during the j^{th} time interval, given the state of its parents $pa(X_n)$ during the j^{th} time intervals;
- $q^{pa(X_m)}_{x^j_m x^{j+1}_m}$ is the parameter associated with the transition from state x^j_m, in which the variable X_m was during the j^{th} time interval, to state x^{j+1}_m, in which the variable X_m will be during the $(j+1)^{th}$ time interval, given the state of its parents $pa(X_m)$ during the j^{th} and the $(j+1)^{th}$ time intervals.

Learning algorithm based on log-likelihood score for the CTNB and inference algorithm for the class of CTBNCs are described in [24].

3 Max-*k* Classifiers

3.1 Definitions

Structural learning for CTBNs is a polynomial time problem with respect to the maximum number of parents k. Nevertheless, increasing k, rapidly brings to considerable computational efforts while implies more data is necessary to learn the node's parameters values conditioned on possible parents' instantiations. To overcome this limitations we propose the following instances from the class of CTBNCs; the Max-k Augmented CTNB (Max-k ACTNB) and the Max-k CTBNC (Max-k CTBNC).

Definition 4. *(Max-k Continuous Time Bayesian Network Classifier). A max-k continuous time Bayesian network classifier is a couple* $\mathcal{M} = \{\mathcal{C}, k\}$, *where* $\mathcal{C}$ *is a continuous time Bayesian network classifier* $\mathcal{C} = \{\aleph, P(Y)\}$ *such that the number of parents* $|Pa(X_n)|$ *for each attribute node* X_n *is bounded by a positive integer* k. *Formally, the following condition holds;* $|Pa(X_n)| \leq k$, $n = 1, 2, ..., N$, $k \geq 0$.

Definition 5. *(Max-k Augmented Continuous Time Naive Bayes). A max-k augmented continuous time naive Bayes classifier is a max-k continuous time Bayesian network classifier such that the class node Y belongs to the parents set of each attribute node X_n, $n = 1, 2, ..., N$. Formally, the following condition holds; $Y \in Pa(X_n)$, $n = 1, 2, ..., N$.*

ACTNB constraints the class variable Y to be a parent of each node X_n, $n = 1, 2, ..., N$. In this way it tries to compensate for relevant dependencies between nodes which could be excluded to satisfy the constraint on the maximum number of parents k.

3.2 Learning

Learning a CTBNC from data consists of learning a CTBN where a specific node, i.e. the class node Y, does not depend on time. In such a case, the learning algorithm runs, for each attribute node X_n, $n = 1, 2, ..., N$, a local search procedure to find its optimal set of parents, i.e. the set of parents which maximizes a given score function. Furthermore, for each attribute node X_n, $n = 1, 2, ..., N$, no more than k parents are selected. The structural learning algorithm proposed in [18] uses a score function (4) based on log-likelihood. This algorithm can be easily adapted to learn a CTBNC by introducing the constraint that the class node Y must not depend on time.

3.3 Log-likelihood and Conditional Log-likelihood

Log-likelihood is not the only scoring function which can be used to learn the structure of a CTBN classifier. Following what presented and discussed in [9], the log-likelihood function:

$$LL(\mathcal{M} \mid \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \log P_{\aleph}(y_i \mid \mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i}) + \log P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i}). \tag{6}$$

consists of two components; $\log P_{\aleph}(y_i \mid \mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$, which measures the classification *capability* of the model, and $\log P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$, which models the dependencies between the nodes.

In [9] the authors remarked that in the case where the number of the attribute nodes X_n, $n = 1, 2, ..., N$ is large, the contribution, to the scoring function value (6), of $\log P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$ overwhelms the contribution of $\log P_{\aleph}(y_i \mid \mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$. However, the contribution of $\log P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})$ is not directly related to the classification accuracy achieved by the classifier. Therefore, to improve the classification performance, in [9] it has been suggested to use the *conditional log-likelihood* as scoring function. In such a case the maximization of the conditional log-likelihood results in maximizing the classification performance of the model without paying specific attention to the discovery of the existing dependencies between the attribute nodes X_n, $n = 1, 2, ..., N$.

The conditional log-likelihood of the CTBNC, which can be written as follows:

$$CLL(\mathcal{M} \mid \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \log P_{\aleph}(y_i \mid \mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i}) = \sum_{i=1}^{|\mathcal{D}|} \log \left(\frac{P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y_i) P_{\aleph}(y_i)}{P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i})} \right)$$
$$= \sum_{i=1}^{|\mathcal{D}|} \log \left(P_{\aleph}(y_i) \right) + \log \left(P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y_i) \right) - \log \left(\sum_{y'} P_{\aleph}(y') P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y') \right). \tag{7}$$

consists of *class probability* (8), *posterior probability* (9), and *denominator* (10) terms. The *class probability* term is estimated from the dataset $\mathcal{D}$ as follows:

$$\sum_{i=1}^{|\mathcal{D}|} \log \left(P_{\aleph}(y_i) \right) = \sum_{y} M[y] \log(\theta_y) \tag{8}$$

where θ_y represents the parameter associated with the probability of class y.

From (5) it is possible to write the following:

$$P_{\aleph}(\mathbf{x}^1, ..., \mathbf{x}^J \mid y) = \prod_{j=1}^{J} q_{x_{m_j} x_{m_j}^{j+1}}^{j\, pa(X_{m_j})} \prod_{n=1}^{N} exp\left(-q_{x_n^j}^{pa(X_n)} \delta_j \right)$$
$$= \prod_{j=1}^{J} q_{x_{m_j}^j}^{pa(X_{m_j})} \theta_{x_{m_j}^j x_{m_j}^{j+1}}^{pa(X_{m_j})} \prod_{n=1}^{N} exp\left(-q_{x_n^j}^{pa(X_n)} \delta_j \right)$$

Therefore, the *posterior probability* term is estimated as follows:

$$\sum_{i=1}^{|\mathcal{D}|} \log \left(P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y_i) \right) = \sum_{n=1}^{N} \sum_{x_n, pa(X_n)} M[x_n \mid pa(X_n)] \log \left(q_{x_n}^{pa(X_n)} \right)$$
$$- q_{x_n}^{pa(X_n)} T[x_n \mid pa(X_n)] + \sum_{x_n' \neq x_n} M[x_n x_n' \mid pa(X_n)] \log(\theta_{x_n x_n'}^{pa(X_n)}). \tag{9}$$

The *denominator* term, because of the sum, can not be decomposed further. The sufficient statistics allow us to write the following:

$$\sum_{i=1}^{|\mathcal{D}|} \log \left(\sum_{y'} P_{\aleph}(y') P_{\aleph}(\mathbf{x}_i^1, ..., \mathbf{x}_i^{J_i} \mid y') \right) = \tag{10}$$

$$= \log \left(\sum_{y'} \theta_{y'} \prod_{n=1}^{N} \prod_{x_n, pa'(X_n)} (q_{x_n}^{pa'(X_n)})^{M[x_n | pa'(X_n)]} \exp(-q_{x_n}^{pa'(X_n)} T[x_n \mid pa'(X_n)]) \right.$$
$$\left. \prod_{x_n' \neq x_n} (\theta_{x_n x_n'}^{pa'(X_n)})^{M[x_n x_n' | pa'(X_n)]} \right)$$

where $pa(X_n) = \{\pi_n \cup y\}$, $pa'(X_n) = \{\pi_n \cup y'\}$, while π_n is the instantiation of the non-class parents of the attribute node X_n.

Unfortunately, no closed form solution exists to compute the optimal value of the model's parameters, i.e. those parameters values which maximize the conditional log-likelihood (7). Therefore, the approach introduced and discussed in [11] is followed. The scoring function is computed by using the conditional log-likelihood, while parameters values are obtained by using the Bayesian approach (3).

4 Numerical experiments

The performance of CTBNCs, namely CTNB, K=2 ACTNB, K=2 CTBNC, K=3 CTBNC, and K=4 CTBNC, is compared to that of DBNs by exploiting synthetic datasets. Classifiers are associated with a suffix related to the scoring function which has been used for learning. Suffix LL is associated with log-likelihood scoring while suffix CLL is associated with conditional log-likelihood scoring. To fairly compare conditional log-likelihood score to log-likelihood score, no graph's structure penalization terms have been added to the two score functions. Numerical experiments for performance estimation and comparison of classifiers are implemented with 10 folds cross validation.

4.1 Synthetic datasets

Accuracy, learning and inference time of different CTBNCs are compared on synthetic datasets generated by sampling from models of increasing complexity. Datasets consist of $1,000$ trajectories with average length ranging from 300 (CTNBs) to 1,400 (K=4 CTBNCs). Analyzed model structures are CTNB, K=2 ACTNB, K=2 CTBNC, K=3 CTBNC, and K=4 CTBNC. For each structure, different assignments of parameters values (q parameters) are sampled in a given interval. Each pair, (*structure*, *parameters assignment*), is used to generate a learning dataset. Performance is analyzed on *full datasets* (100%) and on *reduced datasets*, i.e. when the number and the length of trajectories are reduced to: 80%, 60%, 40%, and 20%. Accuracy values on *full datasets* (100% datasets) are summarized in Table 1 while Figure 2 depicts how the tested models behave when reduced datasets (80%, 60%, 40%, 20%) are used for learning.

DBNs are outperformed by all continuous time models while CLL scoring seems to perform better or at least to be never inferior than LL scoring. Figure 2 shows that CLL scoring strongly outperforms LL scoring when the amount of data is limited. This is probably due to the effectiveness of CLL scoring to discover weak dependencies between variables and thus to its tendency to add the class variable as a parent of all nodes useful for the classification task. On the contrary, when the amount of data is too low, CLL scoring tends to overfit by learning classifiers which are too complex. In these cases, LL scoring achieves poor accuracy too, while the CTNB is the best option (see Fig. 2).

Test	CTNB	$k=2$ ACTNB (LL)	$k=2$ ACTNB (CLL)	$k=2$ CTBNC (LL)	$k=2$ CTBNC (CLL)	$k=3$ CTBNC (LL)	$k=3$ CTBNC (CLL)	$k=4$ CTBNC (LL)	$k=4$ CTBNC (CLL)	DBN-NB1	DBN-NB2
CTNB	**0.95**	**0.95**	0.93	0.93	0.92	0.93	0.82	0.93	0.64	0.80	0.81
K2ACTNB	0.78	0.89	**0.92**	0.76	**0.92**	0.76	0.85	0.76	0.72	0.62	0.63
K2CTBNC	0.68	0.84	**0.86**	**0.85**	**0.86**	**0.85**	0.76	**0.85**	0.60	0.48	0.50
K3CTBNC	0.49	0.65	0.63	0.66	0.63	**0.79**	0.75	**0.79**	0.64	0.32	0.33
K4CTBNC	0.64	0.74	0.79	0.69	0.79	0.76	**0.94**	0.79	0.90	0.40	0.40

Table 1. Classifier's average accuracy value with respect to different categories of the dataset generating model, 10 folds cross validation over *full datasets* (100%). Bolded characters are associated with the best model with 90% of confidence.

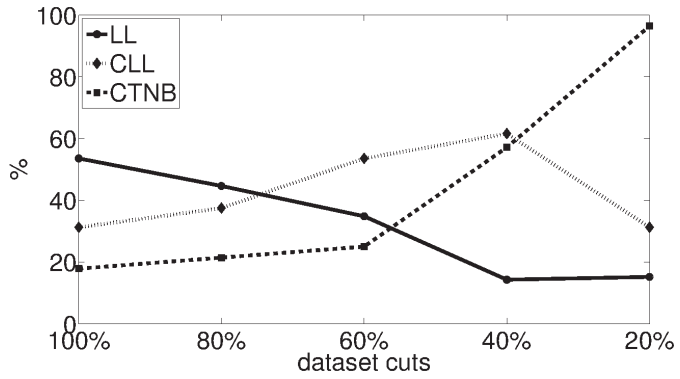


Fig. 2. Percentage of numerical experiments where LL (CLL) achieved a better accuracy value than the one achieved by CLL (LL) with 90% of confidence. Percentage of numerical experiments where CTNB achieved a better/comparable accuracy value than the best accuracy achieved by any continuous time model (bold dotted line) .

Inference times on continuous time models are comparable, while inference time required by DBNs make them impractical. Structural learning of CTBNCs with log-likelihood scoring is slightly faster than with conditional log-likelihood scoring (tables and figures not shown for the sake of brevity).

4.2 Post-stroke rehabilitation dataset

In [26] the authors proposed a movement recognition system to face automatic post-stroke rehabilitation problem. The idea is to provide the patient with a system capable to recognize the movements and to inform him/her about the correctness of the performed rehabilitation exercise. The authors focused on upper limb post-stroke rehabilitation and provided a dataset of 7 rehabilitation exercises. For each exercise 120 multivariate trajectories are recorded by using 29 sensors working with a frequency of 30 Hz [25]. Each movement is addressed separately as classification problem. We focus the attention on 2, and 6 classes problems where classes are associated with the same number of trajectories.

# classes	Measure	CTNB	K=2 ACTNB (LL)	K=2 ACTNB (CLL)	K=2 CTBNC (LL)	K=2 CTBNC (CLL)	K=3 CTBNC (LL)	K=3 CTBNC (CLL)	K=4 CTBNC (LL)	K=4 CTBNC (CLL)
2 classes	Accuracy	0.98	0.97	**0.99**	0.87	0.85	0.87	0.92	0.87	0.95
	Precision	0.97	0.97	**0.99**	0.86	0.85	0.86	0.93	0.86	0.95
	Recall	0.98	0.98	**0.99**	0.88	0.84	0.88	0.92	0.88	0.96
6 classes	Accuracy	**0.91**	**0.91**	**0.89**	0.81	**0.88**	0.81	**0.88**	0.81	**0.88**
	Precision	**0.92**	**0.91**	**0.89**	0.84	**0.89**	0.84	**0.89**	0.84	**0.90**
	Recall	**0.90**	**0.90**	**0.88**	0.81	**0.88**	0.82	**0.88**	0.82	**0.89**

Table 2. Average accuracy, precision and, recall for the post-stroke rehabilitation dataset (10 folds CV). Bolded characters indicate the best models with 90% of confidence.

Accuracy, precision and recall values achieved by almost all CLL classifiers are better than the values achieved by their LL counterparts. For the 6 classes classification problem, in the case where no information about variables' dependency is available, CLL outperforms LL (Table 2). K=2 ACTNB, learned with CLL scoring, implements the optimal trade-off between time and accuracy. Indeed, for both 2 and 6 classes classification problems, the K=2 ACTNB model when learned with CLL, achieves the highest accuracy value and is the fastest to learn, because of the small value of the bound on the number of parents. It is worthwhile to mention that CTBNCs when learned with LL scoring are not capable to solve the 6 classes classification problem for many assignments of the priors values. Indeed, it was necessary to evaluate many priors assignments to achieve an acceptable performance value with LL scoring. On the contrary, CLL scoring seems to be very robust with respect to priors assignment.

5 Conclusions

A conditional log-likelihood scoring function has been developed to learn continuous time Bayesian network classifiers. A learning algorithm for CTBNCs has been designed by combining conditional log-likelihood scoring with Bayesian parameter learning. New classifiers models from the class of CTBNCs have been introduced. Numerical experiments, on synthetic and real world streaming datasets, confirm the effectiveness of the proposed approach for CTBNCs learning. Conditional log-likelihood scoring outperforms log-likelihood scoring and DBNs in terms of accuracy. This behavior becomes more and more evident as the amount of the available streaming data become scarce.

Acknowledgements

The authors would like to acknowledge the many helpful suggestions of the anonymous reviewers which helped to improve the paper clarity and quality. The authors would like to thank Project Automation S.p.A. for funding the Ph.D. programme of Daniele Codecasa.

References

1. Barber, D., Cemgil, A.: Graphical models for time-series. Signal Processing Magazine, IEEE 27(6), 18–28 (2010)
2. Boudali, H., Dugan, J.: A continuous-time bayesian network reliability modeling, and analysis framework. IEEE Trans. on Reliability 55(1), 86–97 (2006)
3. Costa, G., Manco, G., Masciari, E.: Effectively grouping trajectory streams. In: Proc. of the Workshop on New Frontiers in Mining Complex Patterns. pp. 149–151 (2012)
4. Dacorogna, M.: An introduction to high-frequency finance. AP (2001)
5. Dean, T., Kanazawa, K.: A model for reasoning about persistence and causation. Computational Intelligence 5(2), 142–150 (1989)
6. El-Hay, T., Friedman, N., Kupferman, R.: Gibbs sampling in factorized continuous-time markov processes. In: McAllester, D.A., Myllym, P. (eds.) Proc. of the 24th Conf. on UAI. pp. 169–178. AUAI (2008)
7. Fan, Y., Shelton, C.: Sampling for approximate inference in continuous time bayesian networks. In: 10th Int. Symposium on Artificial Intelligence and Mathematics (2008)
8. Fan, Y., Shelton, C.: Learning continuous-time social network dynamics. In: Proc. of the 25th Conf. on UAI. pp. 161–168. AUAI (2009)
9. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning 29(2), 131–163 (1997)
10. Gatti, E., Luciani, D., Stella, F.: A continuous time bayesian network model for cardiogenic heart failure. Flexible Services and Manufacturing Journal pp. 1–20 (2011)
11. Grossman, D., Domingos, P.: Learning bayesian network classifiers by maximizing conditional likelihood. In: Proc. of the 21st Int. Conf. on Machine Learning. pp. 361–368. ACM (2004)
12. Gunawardana, A., Meek, C., Xu, P.: A model for temporal dependencies in event streams. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Advances in Neural Information Processing Systems, pp. 1962–1970 (2011)
13. Langseth, H., Nielsen, T.D.: Latent classification models. Machine Learning 59(3), 237–265 (2005)
14. Masciari, E.: Trajectory clustering via effective partitioning. In: Flexible Query Answering Systems, pp. 358–370. Springer (2009)
15. Nanni, M., Pedreschi, D.: Time-focused clustering of trajectories of moving objects. Journal of Intelligent Information Systems 27(3), 267–289 (2006)
16. Nodelman, U., Koller, D., Shelton, C.: Expectation propagation for continuous time bayesian networks. In: Proc. of the 21st Conf. on UAI. pp. 431–440. Edinburgh, Scotland, UK (July 2005)
17. Nodelman, U., Shelton, C., Koller, D.: Continuous time bayesian networks. In: Proc. of the 18th Conf. on UAI. pp. 378–387. Morgan Kaufmann (2002)
18. Nodelman, U., Shelton, C., Koller, D.: Learning continuous time bayesian networks. In: Proc. of the 19th Conf. on UAI. pp. 451–458. Morgan Kaufmann (2002)
19. Rabiner, L.: A tutorial on hidden markov models and selected applications in speech recognition. Proc. of the IEEE 77(2), 257–286 (1989)
20. Rajaram, S., Graepel, T., Herbrich, R.: Poisson-networks: A model for structured point processes. In: Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics (2005)
21. Saria, S., Nodelman, U., Koller, D.: Reasoning at the right time granularity. In: UAI. pp. 326–334 (2007)

22. Simma, A., Goldszmidt, M., MacCormick, J., Barham, P., Black, R., Isaacs, R., Mortier, R.: Ct-nor: Representing and reasoning about events in continuous time. In: Proc. of the 24th Conf. on UAI. pp. 484–493. AUAI (2008)
23. Simma, A., Jordan, M.: Modeling events with cascades of poisson processes. In: Proc. of the 26th Conf. on UAI. pp. 546–555. AUAI (2010)
24. Stella, F., Amer, Y.: Continuous time bayesian network classifiers. Journal of Biomedical Informatics 45(6), 1108–1119 (2012)
25. Tormene, P., Giorgino, T.: Upper-limb rehabilitation exercises acquired through 29 elastomer strain sensors placed on fabric. release 1.0 (2008)
26. Tormene, P., Giorgino, T., Quaglini, S., Stefanelli, M.: Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. Artificial Intelligence in Medicine 45(1), 11–34 (2009)
27. Truccolo, W., Eden, U., Fellows, M., Donoghue, J., Brown, E.: A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. Journal of neurophysiology 93(2), 1074–1089 (2005)
28. Voit, E.: A First Course in Systems Biology. Garland Science: NY (2012)
29. Xu, J., Shelton, C.: Continuous time bayesian networks for host level network intrusion detection. Machine Learning and Knowledge Discovery in Databases pp. 613–627 (2008)
30. Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. ACM Computing Surveys 38(4) (2006)
31. Zhong, S., Langseth, H., Nielsen, T.D.: Bayesian networks for dynamic classification. Tech. rep., http://idi.ntnu.no/~shket/dLCM.pdf (2012)

Online Batch Weighted Ensemble for Mining Data Streams with Concept Drift

Magdalena Deckert

Institute of Computing Science, Poznań University of Technology,
60-965 Poznań, Poland
magdalena.deckert@cs.put.poznan.pl

Abstract. This paper presents a new framework for dealing with two main types of concept drift (sudden and gradual) in labeled data with decision attribute. The learning examples are processed instance by instance. This new framework, called Online Batch Weighted Ensemble, introduces element of incremental processing into a block-based ensemble of classifiers. Its performance was evaluated experimentally on data sets with different types of concept drift and compared with the performance of Accuracy Weighted Ensemble and Batch Weighted Ensemble. The results show that OBWE improves value of the total accuracy.

1 Introduction

Mining streaming data is one of the recent challenges in data mining. Data streams are characterized by a large amount of data arriving at rapid rate and require efficient processing [8]. Moreover, the data may come from non-stationary sources, where underlying data distribution changes over time. It causes modifications in the target concept definition, which is known as *concept drift* [7]. The main types of changes are usually divided into sudden or gradual concept drifts depending on the rate of changes [11].

Classical static classifiers are incapable of adapting to concept drifts, because they were learned on the out-of-date examples. This is the reason why their predictions become less accurate with time. Some methods have already been proposed to deal with the concept drift problem [7]. They can be divided into two main groups: trigger based and evolving [13]. *Trigger*-based methods use a change detector to identify the occurrence of a change. If the change is detected, then the online classifier, connected with the detector, is re-trained [9]. One of the most popular detectors is DDM described in [6]. On the other hand, *evolving methods* attempt to update their knowledge without explicit information whether the change occurred. An example of such methods is an adaptive ensemble. This paper focuses mainly on block-based ensembles, which component classifiers are constructed on blocks (chunks) of training data. In general, a block-based approach operates in a way that when a new block is available, it is used for evaluation of already existing component and for creation of a new classifier. The new component usually replaces the worst one in the ensemble. For review of those methods see e.g. [7].

The best representative of evolving methods is Accuracy Weighted Ensemble (AWE) proposed in [12]. According to [3, 12], AWE is sensitive to the defined size of a block. Moreover, AWE is very demanding with respect to the memory and time cost, because it builds a new classifier for every incoming block of data. These were motivations for introducing Batch Weighted Ensemble (BWE) in [4]. It incorporates the Batch Drift Detection Method (BDDM) into the AWE inspired structure of the ensemble. Conducted experimental analysis [5] showed that BWE decreases the performance costs, while the total accuracy of classification is held on satisfying level. However, the reaction to the sudden drift, which appears inside the data block is not sufficient—it is delayed until the end of processed block. This was the reason for developing a new environment called Online Batch Weighted Ensemble (OBWE).

The main aim of this paper is to present the new framework for dealing with two main types of concept drift: sudden and gradual drift. This new framework introduces incremental processing of learning instances into the BWE block-based classifier. Its performance was evaluated experimentally on data sets with different types of concept drift and compared with performance of the AWE and standard BWE classifier. Evaluation criteria, on which ensembles will be compared, are: accuracy of classification, use of memory and processing time.

This paper is organized as follows. The next section presents related works on detecting concept drift and block ensembles. Section 3 describes the framework Online Batch Weighted Ensemble. Section 4 is devoted to the experimental evaluation of classifiers for various types of changes. Section 5 concludes this paper.

2 Related Works

This section concentrates on methods that are most related to the presented research study. For reviews of other approaches see [7–9, 11, 13].

First, the Drift Detection Method (DDM) [6] is presented, because it inspired BWE solution. This detector is used in combination with an online classifier. The main idea of DDM is to monitor the error-rate produced by the classifier. For each incoming example the classifier predicts a class label, which is compared with the original (true) one. Classification errors are modeled with a Binomial distribution. When the error increases, it signifies that the data distribution has changed. DDM signals two levels of change: warning and drift according to the sigma rule. When a warning level was exceeded, learning examples are stored in a special buffer. They are collected until drift level is reached. If the alarm level is reached, the previously taught classifier is removed and a new classifier is built from buffer examples. It is possible to observe warning level, followed by a decrease in error rate. This situation is treated as a *false alarm* and the model is not refined. DDM is independent from the learning algorithm and can be easily extended to processing data in blocks.

Evolving methods do not use explicit drift detection. An example of such methods are ensembles of classifiers. They have a natural ability to process data that arrive in blocks. The main idea of the block-based approach is to build

a new classifier for each incoming block of data and then to use this block to evaluate performance of all components existing in the ensemble. Remaining base classifiers receive a weight reflecting their performance. In case when the number of base classifiers is restricted to k, the less accurate one is replaced by the new one. The final answer of the ensemble is constructed by combining single components' votes using weighted majority voting. In AWE classifier, proposed by Wang et al. in [12], each weight w_i of a component is estimated with the formula $w_i = MSE_r - MSE_i$, where MSE_r is mean square error of a random classifier and MSE_i is mean square error of the ith classifier. MSE_r can be calculated as $MSE_r = \sum_c p(c) * (1 - p(c))^2$, where $p(c)$ is the estimation of probability of observing class c in the last block of data. The MSE_i can be expressed by $MSE_i = \frac{1}{|S_n|} \sum_{(x,c) \in S_n} (1 - f_c^i(x))^2$, where S_n is the last block of data and $f_c^i(x)$ is the probability obtained from the classifier i that example x is an instance of class c. In each iteration, k best base classifiers are chosen to form the final ensemble.

According to [12], AWE is sensitive to the chosen block's size. Moreover, due to creating a new classifier in every iteration, AWE has high memory and time requirements. These were motivations for inventing Batch Weighted Ensemble (BWE) in [4]. Next, BWE was improved and fully examined in [5]. The main idea of BWE environment is incorporation of Batch Drift Detection Method (BDDM) into the AWE inspired block-based ensemble. In contrary to typical drift detectors instead of processing instance by instance, BDDM operates on blocks of data. For each example in the block, an accuracy of classification and a standard deviation are calculated incrementally. Next, on the obtained table of accuracy values, a linear regression model is found. It is used to estimate the trend in the data, without finding an ideal adjustment. In the next step, the slope a of the regression model is tested. Value less than 0 means that some change occurred. BDDM distinguishes between two levels of change: warning and drift. If the value of the slope a is less than 0, then default change level is warning. In the end, it is checked whether drift level was obtained. The threshold for the drift was inspired by the DDM [6] and is established using the sigma rule in standardized normal distribution. For more details on BDDM see [4, 5]. Batch Drift Detection Method is incorporated into an ensemble called Batch Weighted Ensemble. BWE operates as follows. In case when the ensemble is empty, a defined number of components is build on bootstrap samples created from the actual block of the data. Alternatively, BWE uses BDDM to check whether the change appeared. If BDDM signals warning or drift level, the weight of every base classifier is computed using appropriate formula. If the maximum size of an ensemble is exceeded, then the base classifier with the lowest weight is removed. If the detector signals drift level, this means that the ensemble must undergo major changes—must be pruned. That is why base classifiers, whose classification accuracy is lower than random guessing, are removed. When all of the ensemble's components are removed, half of them with the highest weights are restored. This is done in order to preserve some of the past knowledge and to avoid learning from scratch. In the end, for every change level, BWE builds a new classifier on

the current block, calculate its weight and adds it to the ensemble. For details on BWE environment see [4, 5]. Experimental results of BWE environment showed the usefulness of incorporating a drift detector inside the block-based adaptive ensembles. Proposed integration reduces computational costs, while the total accuracy of classification is held on satisfying level.

3 Online Batch Weighted Ensemble

Researches on block-based ensembles of classifiers showed that they may insufficiently well react to concept drift appearing inside the data block [10]. They may delay its reaction to the occurring change till the end of actually processed block. This was the reason of developing a new environment called Online Batch Weighted Ensemble (OBWE). The main idea of proposed framework is to introduce an element of incremental processing into the BWE block-based approach. OBWE environment consists of explicit change detector called Online Batch Drift Detection Method and the OBWE ensemble of classifiers.

Algorithm 1: Online Batch Drift Detection Method

Input : C: an ensemble of classifiers; w: weights for current ensemble of classifiers; e: a learning example; r: a regression window size; b: size of the data block
Output: *signal*: flag indicating type of discovered signal

calculate incremental accuracy of classification for example e using ensemble C with weights w;
calculate standard deviation incrementally;
update table containing previous classification accuracies;
if *(regression window size r was exceeded)* **then**
  create regression function on incremental accuracy table;
  if *(a<0)* **then** $\Longleftarrow$ test the slope a of the regression model
    if *(currentAvgAccuracy-currentStdDev<maxAccuracy-3 * maxStdDev)* **then**
      signal = drift;
    else
      signal = warning;
  if *(signal = drift)* **then**
    reset important fields;
if *(block size b was exceeded)* **then**
  store statistics only for the last examples;
Return *signal*

OBDDM modifies the standard BDDM detector in a way, that it processes every single learning example separately. First, the value of accuracy of classification is incrementally updated according to the result of prediction of the ensemble C with weights w for the learning example e. Obtained value is added

to the table with classification accuracies. After every r learning examples, where r is the size of regression window, a linear regression model is found on the whole table with collected accuracies. This shows the tendencies present in the data. The slope a of the regression model less than 0 means that some change exists in the data. If the change was detected, current average value of accuracy is calculated as a mean value from the accuracies stored in the table. Current standard deviation is also obtained from the accuracies table. OBDDM uses the same thresholds for warning and drift levels as the base BDDM detector. After the drift, all important fields and statistics (e.g. accuracy table, current accuracy of classification, maximum values of accuracy and standard deviation) are cleared. In case when the size of a block was exceeded, the statistics calculated for the recent learning examples are stored in order to preserve the information about the trend between two subsequent blocks of data. The pseudo-code of OBDDM is given as Algorithm 1.

OBBDM is incorporated with the OBWE ensemble. Instead of processing streaming data in blocks, it allows reaction after every learning example. However, it maintains a fixed number of recent learning examples as sliding window. The maximum size of the sliding window is restricted to the block size. OBWE operates as follows. When a new learning example is available, it is added to the sliding window. OBWE procedure is executed, if the number of stored learning instances equals the block's size. First, the actual number of component classifiers in the ensemble is checked. If the ensemble is empty, the number of components is constructed on the bootstrap samples achieved from the sliding window. Otherwise, OBDDM is launched in order to check whether the change is present. If OBDDM signals warning level, the weight of every base classifier is computed using formula $w_i = 0.5 * (1 - \frac{e^{6*(x-0.5)} - e^{-6*(x-0.5)}}{e^{6*(x-0.5)} + e^{-6*(x-0.5)}})$. The reason for multiplication by 0.5 is that the codomain of this function is in range $(0; 1)$. Thanks to this all of the existing base classifiers have the same impact in the final answer of the ensemble. Moreover, proposed function w_i, for small values of x, decreases slower. The rate of change is controlled by the multiplication by 6. This value was established empirically by observing the variation of the function w_i—the higher the value is, then the decrease of the function w_i is slower at the beginning and very rapid near the inflection point. For $x = 0.5$ is the inflection point, from which the function w_i decreases faster. The function w_i is symmetrical about the inflection point. The reason for such characteristic is that for warning lever there is no necessity to decrease weights of component classifiers so severely. After weights' update, OBWE checks if the maximum size of an ensemble is exceeded. If so, then the base classifier with the lowest weight is removed. Next, OBWE builds a new classifier on the current window of learning examples, calculate its weight as $w' = maxEnsembleSize - \sum w_j$ and adds it to the ensemble. The newly created classifier obtains such a high weight in order to raise its importance in the final answer of the ensemble. The reason for such assistance is that the new component has the most current knowledge induced from the recent block of data. If the detector signals drift, weights of the existing components are altered with formula $w_i = 0.5 * (1 - \frac{e^{4*(x-0.25)} - e^{-4*(x-0.25)}}{e^{4*(x-0.25)} + e^{-4*(x-0.25)}})$.

Algorithm 2: Online Batch Weighted Ensemble

Input : S: a data stream of examples; b: size of the data block; r: size of the regression window; bsC: number of bootstrap classifiers; $maxC$: maximum number of classifiers in ensemble; C: a set of previously trained classifiers

Output: C: an updated set of classifiers; w: an updated weights for current ensemble of classifiers

foreach *(learning example $s_i \in S$)* **do**
- add example s_i to the learning window;
- **if** *(size of learning window = size of a block b)* **then**
  - **if** *(size of ensemble = 0)* **then**
    - **foreach** *(j = 1 .. bsC)* **do**
      - train classifier C_j on $bootstrapSample(window)$;
      - $C \leftarrow C \cup C_j$;
      - $w_j = maxC/bsC$;
  - **else**
    - `OBDDM` (C, w, s_i, r, b); {build Online Batch Drift Detection Method}
    - **if** *(signal=warning)* **then**
      - **foreach** *(classifier $C_j \in C$)* **do**
        - compute w_j using a formula:
        - $w_j = 0.5 * (1 - \frac{e^{6*(x-0.5)} - e^{-6*(x-0.5)}}{e^{6*(x-0.5)} + e^{-6*(x-0.5)}})$;
      - **if** *(memory buffer is full)* **then**
        - remove classifier with the lowest weight;
      - train classifier C' on current learning window;
      - $C \leftarrow C \cup C'$;
      - compute weight w' of classifier C' as:
      - $w' = maxEnsembleSize - \sum w_j$;
    - **else if** *(signal=drift)* **then**
      - **foreach** *(classifier $C_i \in C$)* **do**
        - compute w_j using a formula:
        - $w_j = 0.5 * (1 - \frac{e^{4*(x-0.25)} - e^{-4*(x-0.25)}}{e^{4*(x-0.25)} + e^{-4*(x-0.25)}})$;
      - **if** *(memory buffer is full)* **then**
        - remove classifier with the lowest weight;
      - **foreach** *(classifier $C_j \in C$)* **do**
        - **if** *($w_j <= \frac{1}{classes_{no}}$)* **then**
          - remove classifier C_j;
      - **if** *(size of ensemble = 0)* **then**
        - restore half of the best classifiers with their weights;
      - train classifier C' on current learning window;
      - $C \leftarrow C \cup C'$;
      - compute weight w' of classifier C' as:
      - $w' = maxEnsembleSize - \sum w_j$;
  - remove the oldest example from learning window;

Return C

All of the parameters' values were also established empirically by observing the variation of the function w_i. Proposed function for recalculating weights has the inflection point for $x = 0.25$. Additionally, thanks to multiplication by 4, it decreases faster than the one for warning. The reason for this behavior is when the sudden drift is detected the components must be punished more quickly and severely for their mistakes. Next, OBWE prunes the ensemble—the base classifiers, whose accuracy of classification is less than $\frac{1}{|classes|}$, are removed. In case when all of the ensemble's components are removed, half of them with the highest weights are restored. Then, OBWE builds a new classifier on the current window of learning examples, calculate its weight as $w' = maxEnsembleSize - \sum w_j$ and adds it to the ensemble. In the end, OBWE removes only one—the oldest learning example from the stored sliding window. The pseudo-code of OBWE is given as Algorithm 2.

4 Experimental Evaluation

Three different classifiers were chosen for experimental comparison: two block-based approaches AWE, BWE with BDDM and OBWE environment. All classifiers were implemented in Java and were embedded into the Massive Online Analysis framework for mining streaming data. More about the MOA project can be found in [1] and at the website [1]. All base classifiers were constructed using the C4.5 decision tree algorithm (WEKA's J48)—to be consistent with the related works [5, 12]. Unpruned version of the tree was used in order to obtain a more precise description of the current block. Thanks to this the component classifiers will reflect only knowledge obtained from one block of data, so they will be more specialized for different knowledge regions.

Only one block size equals 1000 was tested. However, three different sizes of regression window were checked: 10, 100 and 1000. To estimate classification performance the EvaluateInterleavedTestThanTrain method from MOA was used. It first uses each example in the stream to assess the classification accuracy and than this example is used to re-train (update) the classifier. Evaluation measures were recorded after every 100 examples. Besides the total classification accuracy also values of accumulated processing time (from the beginning of the learning phase) and the size of current model (expressed by memory size) were logged.

All experiments were carried out on datasets with different types of changes, such as gradual drifts, sudden changes, complex (mixed) changes, blips (representing rare events, which should not be treated as real drifts) and no drifts (for which a classifier should not be updated). Nine different datasets were used: three real datasets, which are often considered by other researchers and six artificial datasets obtained using MOA generators, which precise descriptions can be found in MOA Manual [2]. Detailed characteristics of the datasets are given in Table 1.

Due to the page limits only the most representative results are presented. All compared algorithms were evaluated in terms of classification accuracy, memory

[1] see: http://moa.cs.waikato.ac.nz/

Table 1. Characteristics of datasets

Dataset	Examples	Attributes	Classes	Change type	Parameters
CovType	581012	54	7	unknown	N/A
Electricity	45312	8	2	unknown	N/A
Poker	829201	11	10	unknown	N/A
Hyperplane	100000	10	4	gradual	slow change: t=0.001
RBFGradual	100000	20	4	gradual	p=5001, a=45, w=1000
STAGGER	100000	3	2	sudden	p=3001, a=90, w=1
RBFSudden	100000	20	4	sudden	p=5001, a=90, w=1
RBFBlips	100000	20	4	blips	p=24990, a=80, w=200
RBFNoDrift	100000	10	2	N/A	default

usage and total processing time. The accuracy values and memory were averaged over recorded time points. The values of all measures on datasets are presented in Tables: 2, 3 and 4. They will be interpreted in the next section. For better insight into dynamics of learning figures after processing every learning example were plotted. Again, due to the space limits only the representative figures are presented—see Figures: 1, 2 and 3.

Dataset	AWE	BWE	OBWE-R10	OBWE-R100	OBWE-R1000
CovType	81,52	82,60	**85,54**	83,06	82,13
Electricity	73,53	71,41	74,20	**75,67**	71,77
Poker	78,32	75,49	81,18	**82,11**	77,12
Hyperplane	70,91	77,11	78,13	**81,90**	77,87
RBFGradual	75,25	74,49	82,68	**84,16**	78,31
STAGGER	**78,30**	**78,30**	77,65	74,87	77,36
RBFSudden	75,37	74,40	**83,68**	82,67	78,18
RBFBlips	88,41	85,55	87,86	**89,12**	85,90
RBFNoDrift	88,01	87,41	86,27	**88,22**	87,31

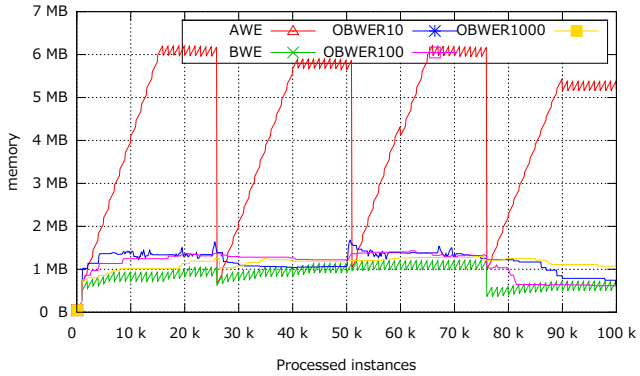
Table 2. Average values of classification accuracy [%]

Dataset	AWE	BWE	OBWE-R10	OBWE-R100	OBWE-R1000
CovType	5,49	0,79	1,12	1,19	1,13
Electricity	0,76	0,58	0,82	0,79	0,68
Poker	1,48	1,21	1,47	1,45	1,31
Hyperplane	0,63	1,06	1,28	1,25	1,21
RBFGradual	1,40	0,42	1,17	1,20	0,70
STAGGER	0,50	0,07	0,18	0,17	0,15
RBFSudden	1,40	0,43	1,11	1,13	0,68
RBFBlips	4,13	0,82	1,13	1,08	1,09
RBFNoDrift	4,02	0,79	1,02	0,99	0,87

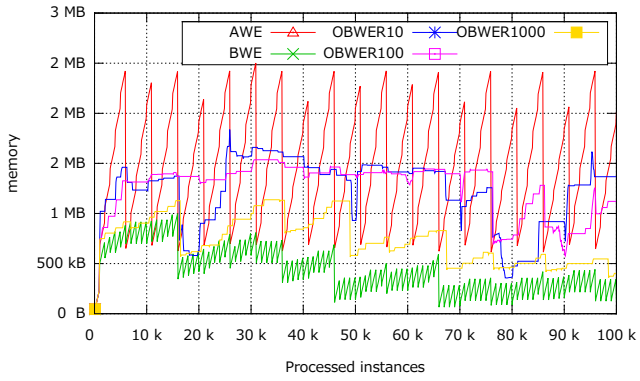
Table 3. Average amounts of used memory [MB]

Dataset	AWE	BWE	OBWE-R10	OBWE-R100	OBWE-R1000
CovType	897,01	338,30	837,51	258,87	163,35
Electricity	20,83	11,23	30,61	15,12	11,09
Poker	629,35	287,56	617,83	380,71	290,04
Hyperplane	35,74	37,27	201,52	54,05	37,67
RBFGradual	68,00	20,34	92,34	48,36	23,51
STAGGER	33,09	3,96	18,55	7,74	6,02
RBFSudden	68,50	20,69	118,84	46,82	23,79
RBFBlips	188,64	31,54	215,64	49,03	33,48
RBFNoDrift	228,14	28,25	201,55	39,25	27,21

Table 4. Total processing time [s]



(a) RBFBlips



(b) RBFGradual

Fig. 1. Memory usage for selected datasets

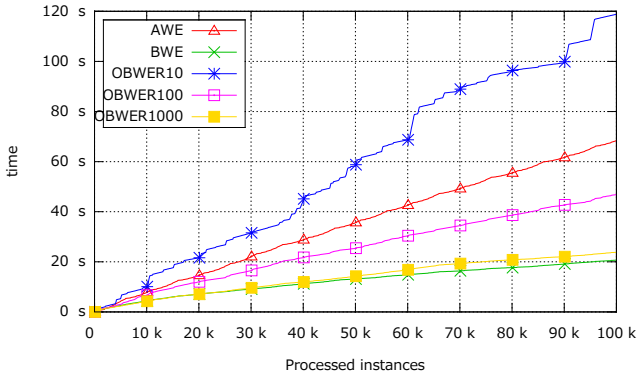
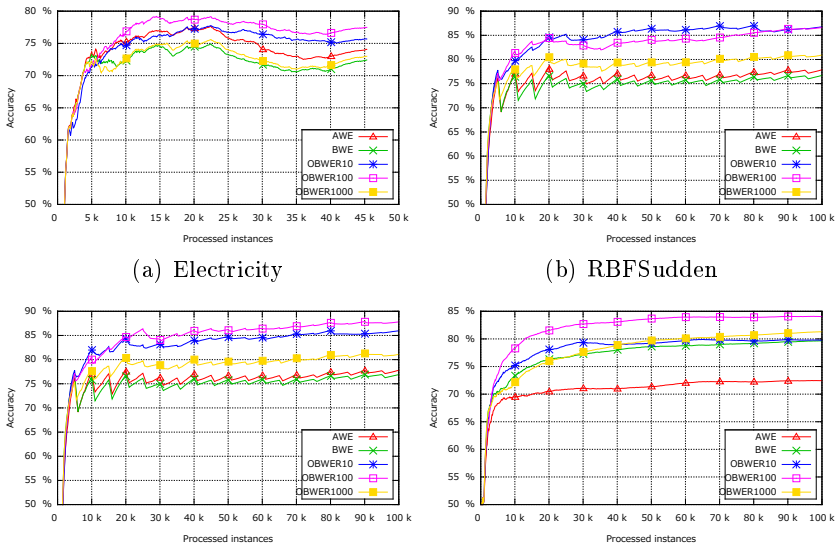


Fig. 2. Processing time for selected dataset



(a) Electricity (b) RBFSudden

(c) RBFGradual (d) Hyperplane

Fig. 3. Classification accuracy for selected datasets

5 Discussion of Results and Final Remarks

In this paper, a new framework for dealing with two main types of concept drift: sudden and gradual drift was presented. This framework, called Online Batch Weighted Ensemble, introduces incremental processing of learning instances into the BWE block-based environment.

After comparing results of total accuracy of classification, one can notice that OBWE obtains the highest value of this measure. The reason of this behavior is that, thanks to incremental processing, OBWE can react to the change more quickly. Instead of waiting until the end of the block, it can re-train immediately after the change was detected. Moreover, in most of the cases, OBWE with the size of regression window equals 100 is the best with respect to the total accuracy of classification. The second position belongs to OBWE with the size of regression window equals 10. AWE achieves the average value of classification accuracy. The worst, with respect to the accuracy of classification, are OBWE with the size of regression window equals 1000 and the standard BWE environment.

Results obtained on memory showed that AWE uses the most amount of memory. The standard BWE environment have the lowest memory requirements. OBWE, in comparison to the standard BWE, needs more memory, however it is still smaller amount than AWE demands. Moreover, in most of the cases, the lower the size of regression window is, the higher size of memory is needed.

Comparison of processing time showed that the standard BWE environment is the fastest classifier. OBWE with regression window size 1000 works for similar period of time as BWE. OBWE with regression size 100 operates longer than the standard BWE environment but is much faster than AWE and OBWE with regression size 10. Those two classifiers are the slowest ones. In more than a half of the cases, OBWE with regression window size 10 operates the longest and less time needs AWE classifier. In case when AWE is the slowest one, then OBWE with regression window size 10 works only a little bit faster than AWE.

In majority of cases, the type of change existing in the dataset does not influence the obtained results. However, for dataset with blips and when there does not exist any change, the advantage of BWE and OBWE over AWE is visible in memory usage—AWE is 4 to 5 times more demanding.

To sum up, the experimental evaluation on nine data sets with different types of drift showed that OBWE improves reaction to the drift, which results in higher classification accuracy. On the other hand, incremental processing is more demanding with respect to the evaluation measures like memory usage and processing time. Moreover, experiments showed that it is unprofitable to highly decrease the size of regression window. The performance requirements rise with decreasing size of regression window but the gain on accuracy of classification is not so significant comparing to the average regression window size 100.

The future research may consider integration of the proposed Online Batch Weighted Ensemble environment with an incremental learning algorithm e.g. Very Fast Decision Trees (VFDT).

References

1. Bifet A., Holmes G., Kirkby R., Pfahringer B.: MOA: Massive Online Analysis., Journal of Machine Learning Research (JMLR), vol. 11, pp. 1601-1604, 2010.
2. Bifet A., Kirkby R.: Massive Online Analysis Manual., COSI, 2009.
3. Brzezinski D., Stefanowski J: Accuracy updated ensemble for data streams with concept drift. Proceedings of HAIS Part II, LNAI, vol. 6679, pp. 155-163, 2011.
4. Deckert M.: Batch Weighted Ensemble for Mining Data Streams with Concept Drift., Proceedings of the 19th International Conference on Foundations of Intelligent Systems (ISMIS 2011), Warsaw, Poland, LNCS, vol. 6804, pp. 290-299, 2011.
5. Deckert M., Stefanowski J.: Comparing Block Ensembles for Data Streams with Concept Drift, New Trends in Databases and Information Systems, Advances in Intelligent Systems and Computing, vol. 185, pp. 69-78, 2012.
6. Gama J., Medas P., Castillo G. and Rodrigues P.: Learning with Drift Detection., In SBIA Brazilian Symposium on Artificial Intelligence, LNAI, vol. 3171, pp. 286-295, 2004.
7. Gama J.: Knowledge Discovery from Data Streams., CRC Publishers 2010.
8. Kuncheva L.I.: Classifier ensembles for changing environments., In Proceedings of the 5th International Workshop on Multiple Classifier Systems (MCS2004), Italy, LNCS, vol. 3077, pp. 1-15, 2004.
9. Kuncheva L.I.: Classifier ensembles for detecting concept change in streaming data: Overview and perspectives., In Proceedings of the 2nd Workshop SUEMA 2008 (ECAI 2008), Greece, pp. 5-10, 2008.
10. Nishida K., Yamauchi K., Omori T.: ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments., Multiple Classifier Systems, LNCS, vol. 3541, pp. 176-185, 2005.
11. Tsymbal A.: The problem of concept drift: Definitions and related work. Technical Report, Trinity College, Dublin, Ireland, 2004.
12. Wang H., Fan W., Yu P.S., Han J.: Mining concept-drifting data streams using ensemble classifiers., In Proceedings of the ACM SIGKDD, pp. 226-235, 2003.
13. Zliobaite I.: Learning under Concept Drift: an Overview. Technical Report, Vilnius University, Lithuania, 2009.

A Relational Unsupervised Approach to Author Identification

F. Leuzzi[1], S. Ferilli[1,2], and F. Rotella[1]

[1] Dipartimento di Informatica – Università di Bari
{fabio.leuzzi, stefano.ferilli, fulvio.rotella}@uniba.it
[2] Centro Interdipartimentale per la Logica e sue Applicazioni – Università di Bari

Abstract. In the last decades speaking and writing habits have changed. Many works faced the author identification task by exploiting frequentist approaches, numeric techniques or writing style analysis. Following the last approach we propose a technique for author identification based on First-Order Logic. Specifically, we translate the complex data represented by natural language text to complex (relational) patterns that represent the writing style of an author. Then, we model an author as the result of clustering the relational descriptions associated to the sentences. The underlying idea is that such a model can express the typical way in which an author composes the sentences in his writings. So, if we can map such writing habits from the unknown-author model to the known-author model, we can conclude that the author is the same. Preliminary results are promising and the approach seems practicable in real contexts since it does not need a training phase and performs well also with short texts.

1 Introduction

Speaking and writing habits have changed in the last decades, and many works have investigated the author identification task by exploiting frequentist approaches, numeric techniques and writing style analysis. The spreading of documents across the Internet made the writing activity faster and easier compared to past years. Thus, author identification became a primary issue, due to the increasing number of plagiarism cases. In order to face such problems, several approaches have been attempted in the Machine Learning field [1, 4, 12, 18].

The authorship attribution task is well-understood (given a document, determine who wrote it) although amenable to many variations (given a document, determine a profile of the author; given a pair of documents, determine whether they were written by the same author; given a document, determine which parts of it were written by a specific person), and its motivation is clear. In applied areas such as law and journalism knowing the author's identity may save lives.

The most common approach for testing candidate algorithms is to cast the problem as a text classification task: given known sample documents from a small, finite set of candidate authors, assess if any of those authors wrote a questioned document of unknown authorship. A more lifelike approach is: given

a set of documents by a single author and a questioned document, determine whether the questioned document was written by that particular author or not. This is more interesting for professional forensic linguistics because it is a primary need in that environment.

This setting motivated us to face the following task: given a small set (no more than 10, possibly just one) of "known" documents written by a single person and a "questioned" document, determine whether the latter was written by the same person who wrote the former.

Performing a deep understanding of the author to seize his style is not trivial, due to the intrinsic ambiguity of natural language and to the huge amount of common sense and linguistic/conceptual background knowledge needed to switch from a purely syntactic representation to the underlying semantics. Traditional approaches are not able to seize the whole complex network of relationships, often hidden, between events, objects or a combination of them. Conversely relational approaches treat natural language texts as complex data from which mining complex patterns.

So after extracting and making explicit the typed syntactical dependencies of each sentence, we formally express them in a First-Order Logic representation. In this way the unstructured texts in natural language are expressed by complex (relational) patterns on which automatic techniques can be applied. Exploiting such patterns, the author's style can be modelled in order to classify a new document as written by the same author or not.

For the sake of clarity, from now on we refer to the known author used for training as the *base*, and to the unknown author that must be classified as *target*.

This work is organized as follows: the next section describes related works; Section 3 outlines the proposed approach, that is evaluated subsequently. Lastly, we conclude with some considerations and future works.

2 Related Work

There is a huge amount of research conducted on Author Identification in the last 10 years. With the spread of anonymous documents in Internet, authorship attribution becomes important. Researches focus on different properties of texts, the so-called *style markers*, to quantify the writing style under different labels and criteria. Five main types of features can be found: lexical, character, syntactic, semantic and application specific. The lexical and character features consider a text as a mere sequence of word-tokens or characters, respectively. An example of the first category is [2] in which new lexical features are defined for use in stylistic text classification, based on taxonomies of various semantic functions of certain choice words or phrases. While this work reaches interesting results, it is based on the definition of arbitrary criteria (such as the 675 lexical features and taxonomies) and requires language-dependent expertise.

In [20] the authors build a suffix tree representing all possible character n-grams of variable length and then extract groups of character n-grams as features. An important issue of such approaches based on character feature is the

choice of n, because a larger n captures more information but increases the dimensionality of the representation. On the other hand, a small n might not be adequate to learn an appropriate model.

Syntactic features are based on the idea that authors tend to unconsciously use similar syntactic patterns. Therefore they exploit information such as PoS-tags, sentence and phrase structures. In addition to the need of robust and accurate NLP tools to perform syntactic analysis of texts, a major drawback of these approaches is the huge amount of feature extracted they require (e.g., in [19] there are about 900k features).

Semantic approaches rely on semantic dependencies obtained by external resources, such as taxonomies or thesauri. In [13] the authors exploits WordNet[5] to detect "semantic" information between words. Although the use of an external taxonomic or ontological resource can be very useful for these purposes, such resources are not always available and often do not exist at all for very specific domains.

Finally, there are non-general-purpose approaches, that define application-specific measures to better represent the style in a given text domain. Such measures are based on the use of greetings and farewells in the messages, types of signatures, use of indentation, paragraph length, and so on [11].

While the various approaches faced the problem from different perspectives, a common feature to all of them is their using a flat (vectorial) representation of the document/phrases. Even the two before the last approach, although starting from syntactic trees or word/concept graphs, subsequently create new flat features, losing in this way the relations embedded in the original texts.

A different approach that preserves the phrase structure is presented in [15]. In this work a probabilistic context-free grammar (PCFG) is built for each author and then each test document is assigned to the author whose PCFG produced the highest likelihood for such a document. While this approach takes into account the syntactic tree of the sentences, it needs of many documents per author to learn the right probabilities. Thus it is not applicable in settings in which a small set of documents of only one author is available. Moreover we believe that the exploitation of only parse trees is not enough to characterize the author's style, conversely it should be better to enrich the syntactical relationships with grammatical ones.

Differently from all of these, our approach aims at preserving the informative richness of textual data by extracting and exploiting complex patterns from such complex data.

3 Proposed Approach

Natural Language Text is a complex kind of data encoding implicitly the author's style. We propose to translate textual data into a relational description in order to make explicit the complex patterns representing the author's style. The relational descriptions are clustered using the similarity measure presented in [6], where the threshold to be used as a stopping criterion is automatically

recognized. We apply this technique to build both base and target models. Then, the classification results from the comparison of these two models. The underlying idea is that the target model describes a set of ways in which the author composes the sentences. If we can bring back such writing habits to the base model, we can conclude that the author is the same.

3.1 The representation formalism

Natural language texts are processed by ConNeKTion [9] (acronym for 'CONcept NEtwork for Knowledge representaTION'), a framework for conceptual graph learning and exploitation. This framework aims at partially simulating some human abilities in the text understanding and concept formation activity, such as: extracting the concepts expressed in given texts and assessing their relevance [7]; obtaining a practical description of the concepts underlying the terms, which in turn would allow to generalize concepts having similar description [16]; applying some kind of reasoning 'by association', that looks for possible indirect connections between two identified concepts [10]; identifying relevant keywords that are present in the text and helping the user in retrieving useful information [17].

In this work we exploit ConNeKTion in order to obtain a relational representation of the syntactic features of the sentences. In particular exploiting the *Stanford Parser* and *Stanford Dependencies* tools [8, 3] we obtain phrase structure trees and a set of grammatical relations (typed dependencies) for each sentence. These dependencies are expressed as binary relations between pairs of words, the former of which represents the governor of the grammatical relation, and the latter its dependent. Words in the input text are normalized using lemmatization instead of stemming, which allows to distinguish their grammatical role and is more comfortable to read by humans. ConNeKTion also embeds JavaRAP, an implementation of the classic Anaphora Resolution Procedure [14]. Indeed, the subject/objects of the sentences are often expressed as pronouns, referred to the first occurrence of the actual subject/object. After applying all these pre-processing steps, we translate each sentence into a relational pattern. In particular, each sentence is translated into a Horn Clause of the form:

$$sentence(IdSentence) \text{ :- } description(IdSentence).$$

where $description(Idsentence)$ is a combination of the following atoms which reflect the relations between the words in the sentence:

- $phrase(Tag, IdSentence, Pos)$ represents a constituent whose Tag is the type of phrase (e.g. NP, VP, S,...) and Pos is the term position in the phrase;
- $term(IdSentence, Pos, Lemma, PosTag)$ defines a single term whose position in the sentence is Pos, its lemma is $Lemma$ and its part-of-speech (e.g. N,V,P,...) is $PosTag$;
- $sd(IdSentence, Type, PosGov, PosDep)$ represents the grammatical relation $Type$ (e.g. dobj, subj,...) between the governor word in position $PosGov$ and the dependent word in position $PosDep$.

This allows us to represent all the relationships between the terms, their grammatical relations and the phrases to which they belong.

3.2 The similarity measure

The similarity strategy exploited here was presented in [6]. It takes values in $]0, 4[$ and is computed by repeated applications of the following formula to different parameters extracted from the relational descriptions:

$$sf(i', i'') = sf(n, l, m) = \alpha \frac{l+1}{l+n+2} + (1-\alpha)\frac{l+1}{l+m+2}$$

where:

- i' and i'' are the two items under comparison;
- n represents the information carried by i' but not by i'';
- l is the common information between i' and i'';
- m is the information carried by i'' but not by i';
- α is a weight that determines the importance of i' with respect to i'' (0.5 means equal importance).

More precisely, the overall similarity measure carries out a layered evaluation that, starting from simpler components, proceeds towards higher-level ones repeatedly applying the above similarity formula. At each level, it exploits the information coming from lower levels and extends it with new features. At the basic level terms (i.e., constants or variables in a Datalog setting) are considered, that represent objects in the world and whose similarity is based on their properties (expressed by unary predicates) and roles (expressed by their position as arguments in n-ary predicates). The next level involves atoms built on n-ary predicates: the similarity of two atoms is based on their "star" (the multiset of predicates corresponding to atoms directly linked to them in the clause body, that expresses their similarity 'in breadth') and on the average similarity of their arguments. Since each of the four components ranges into $]0, 1[$, their sum ranges into $]0, 4[$. Then, the similarity of sequences of atoms is based on the length of their compatible initial subsequence and on the average similarity of the atoms appearing in such a subsequence. Finally, the similarity of clauses is computed according to their least general generalization, considering how many literals and terms they have in common and on their corresponding lower-level similarities.

3.3 Building models

Obtained a relational description for each sentence as described in Section 3.1, we applied the similarity measure described in Section 3.2 to pair of sentences. In particular, for each training-test couple we computed an upper triangular similarity matrix between each pair sentences. As can be seen in Figure 1 the global matrix can be partitioned into three parts, the top-left submatrix (filled with diagonal lines) contains the similarity scores between each pair of sentences of known documents (base). The bottom-right one (filled with solid grey) includes the similarities between pairs of sentences belonging to the unknown document (target). The top right submatrix reports the similarity scores across known and unknown documents.

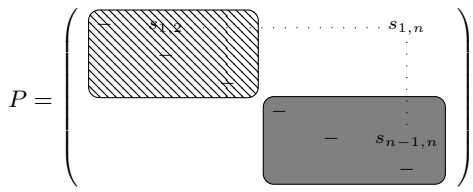


Fig. 1. Global similarity matrix. Each $s_{i,j}$ represents the similarity between the sentence i and j calculated as explained in Section 3.2

Then, we performed an agglomerative clustering to both base and target submatrices according to Algorithm 1. Initially each description makes up a different singleton cluster; then the procedure works by iteratively finding the next pair of clusters to be merged according to a *complete link* strategy. Complete link states that *the distance of the farthest items of the involved clusters must be less than a given threshold*. In this work we refer to a *model* as a possible grouping of similar descriptions, as obtained by running the clustering algorithm with a given threshold.

In this perspective, there is the need of establishing the threshold by which pairwise clustering is carried out for each *model*. Our approach is based on the idea that as long as the threshold increases, also the number of clusters grows, and thus the merging becomes more and more difficult. So, we consider as cut point the greatest gap between the number of clusters obtained with a threshold and the next one obtained by performing clustering with a greater threshold. It is easy to note that a given difference value obtained with many clusters is less significant than the one obtained with a smaller number of clusters.

Taking into account such considerations we have defined the following function that encodes such intuitive assumptions. Given a sequence of models $< m_1, ..., m_n >$ obtained by repeating the clustering procedure with the increment of the cut-threshold by step 0.05, and $c(m_i)$ that computes the number of clusters in the i-th model, we can define:

$$g(i) = \frac{c(m_{i+1})}{c(m_i)} \quad \text{and} \quad th_i = \arg\max_i g(i)$$

where $0 \leq i < n$ and th_i is the desired threshold associated to the model m_i yielding the greatest distance from the model m_{i+1} (see Algorithm 2). Since our similarity measure ranges in $]0, 4[$, the thresholds varies within such range.

Chosen the appropriate thresholds, we defined base and target models and thus we performed the classification. In particular, as can be seen in Algorithm 3, for each cluster in the target model having more than one item, if it can be merged with at least one cluster in the base model (under the complete link assumption), the author is the same, otherwise it is not. Such merging check exploits the top right submatrix. Moreover it uses the maximum threshold between base and target model, which making harder a full alignment between target and base clusters and ensures more precision in the classifications.

Algorithm 1 Relational pairwise clustering.
Interface: *pairwiseClustering(M,T)*.

Input: M is the similarity matrix; T is the threshold for similarity function.
Output: set of clusters.

```
pairs ← empty
averages ← empty
for all item : M.rows do
  newCluster ← item
  clusters.add(newCluster)
end for
for all pair(C_k, C_z) | C_k, C_z ∈ clusters do
  if completeLink(M, C_k, C_z, T) then
    pairs.add(C_k, C_z)
    averages.add(getScoreAverage(C_k, C_z))
  end if
end for
pair ← getBestPair(pairs, averages)
merge(pair)
return clusters
```

completeLink(matrix,cluster$_1$,cluster$_2$,threshold) → TRUE if complete link assumption for the passed clusters holds, FALSE otherwise.
getBestPair(pairs,averages) → returns the pair having the maximum average.

4 Evaluation

We evaluated our procedure using the training set provided in the 9th evaluation lab on uncovering plagiarism, authorship, and social software misuse (PAN) held as part of the CLEF 2013 conference. Although this challenge has already taken place, we could not compare our outcomes with the official challenge results because the test set is not yet publicly available. However, since our approach does not require a training phase, we were able to exploit the training set for this purpose. Such a dataset is composed by 10 problems for the English language, 20 problems for Greek and 5 problems for Spanish. In this evaluation we will consider the English problems only, since the current version of ConNeKTion is based on the Stanford NLP tools, that cannot deal with the other two languages. However, our approach can be easily extended to the other languages, as long as suitable NLP tools for them are available.

Table 1 reports, for each problem (whose ID, as indicated in the original data set, is reported in the first column), information both on the documents written by the base author and on the unknown document written by the target author. As to the former, it specifies the number of documents, the number of clauses generated by the sentences they contain and their average length. As to the latter, it shows the number of corresponding clauses/sentences and their average length. It also reports the experimental outcomes including the expected

Algorithm 2 Best model identification.
Interface: *getBestModel(M, T_{lower}, T_{higher})*

Input: M is the similarity matrix; T_{lower} is the starting threshold, T_{higher} is the maximum threshold that can be attempted.
Output: *bestModel* that is the model having the best threshold.

```
t ← T_lower
models ← ∅
thresholds ← ∅
for all t < T_higher do
  clusters ← pairwiseClustering(M)
  models.add(clusters)
  thresholds.add(t)
  t ← t + 0.5
end for
maxHop ← 0
bestModel ← null
for all m_i | m_i ∈ models, i > 0 do
  hop ← (m_i.size * 100)/m_{i-1}.size
  if maxHop < hop then
    maxHop ← hop
    bestModel ← m_{i-1}
  end if
end for
return bestModel
```

class, the class predicted by our approach and a score computed as the number of aligned cluster across models over the total number of target-model clusters. The average number of sentences for the known and unknown documents is respectively 164.9 and 56.2, and the average lengths of the descriptions are respectively of 176.01 and 203.97. It can be noted that on average the known documents consist of many short sentences while the unknown documents are composed of few long sentences. A singular situation happens in problem 'EN23' where summing all the sentences of the known documents we obtain half of the sentences belonging to the corresponding unknown one.

In our experimentation we reported the same measures required in the original challenge. Precision and Recall (and, consequently, F_1 measure) are equal to 0.7 in the case of a 'hard' classification obtained considering as Yes only the scores equal to 1.0. Softening the classification threshold to 0.9, these statistics become 0.8 for all metrics. In fact, it is worth noting that the correct classification of the problem 'EN23' was not reached for just 0.08, but this is the problem with the smallest number of clauses to be clustered, hence we could hypothesize that a sufficiently refined model has not been reached for a while.

The complete quantitative results of the PAN 2013 challenge (English section) are reported in Table 2. In this table are shown the nick names of the challenge participants with the performances of their systems. Considering that

Algorithm 3 Complete classification procedure.

Input: O_{known} is the set of descriptions (represented as in Section 3.1) obtained from the known-author documents; $O_{unknown}$ is the set of descriptions obtained from the unknown-author document; T_{lower} is the starting threshold, T_{higher} is the maximum threshold that can be attempted.
Output: Classification outcome.

$M_{known} \leftarrow getSimilarities(O_{known})$
$model_{known} \leftarrow getBestModel(M_{known}, T_{lower}, T_{higher})$
$M_{unknown} \leftarrow getSimilarities(O_{unknown})$
$model_{unknown} \leftarrow getBestModel(M_{unknown}, T_{lower}, T_{higher})$
$t \leftarrow max(t_{known}, t_{unknown})$
$O \leftarrow O_{known}$
$O.add(O_{unknown})$
$M \leftarrow getSimilarities(O)$
$class \leftarrow true$
for all $(C_k, C_u) \mid C_k \in model_{known} \wedge C_u \in model_{unknown}$ **do**
  if $!completeLink(M, C_k, C_u, t)$ **then**
    $class \leftarrow false$
  end if
end for
return $class$

completeLink(matrix,cluster$_1$*,cluster*$_2$*,threshold)* $\rightarrow$ TRUE if complete link assumption for the passed clusters holds, FALSE otherwise.
getSimilarities(list) $\rightarrow$ returns the similarity matrix between all pairs of objects in 'list'.

the performance of the winner approach in PAN 2013 has reached an F_1 measure equal to 0.8, these preliminary results can be considered very promising and motivate us to further proceed in this research direction.

5 Conclusions

This work proposes a technique for author identification based on First-Order Logic. It is motivated by the assumption that making explicit the typed syntactical dependencies in the text one may obtain significant features on which basing the predictions. Thus, this approach translates the complex data represented by natural language text to complex (relational) patterns that allow to model the writing style of an author. Then, these models can be exploited to classify a novel document as written by the author or not. Our approach consists in translating the sentences into relational descriptions, then clustering these descriptions (using an automatically computed threshold to stop the clustering procedure). The resulting clusters represent our model of an author. So, after building the models of the base (known) author and the target (unknown) one, the comparison of these models suggests a classification (i.e., whether the target author is the same

Table 1. Dataset details and outcomes

ID	Known docs			Unknown doc		Outcomes		
	$\#_{docs}$	$\#_{clauses}$	μ_{length}	$\#_{clauses}$	μ_{length}	Expected	Class	Score
EN04	4	261	121.06	62	136.60	Y	Y	1.0
EN07	4	260	121.48	44	195.47	N	Y	1.0
EN11	2	109	185.87	39	160.41	Y	Y	1.0
EN13	3	109	156.99	65	134.65	N	N	0.6
EN18	5	274	154.25	53	165.49	Y	Y	1.0
EN19	3	139	164.35	56	210.05	Y	N	0.37
EN21	2	109	210.89	24	269.21	N	N	0.67
EN23	2	51	217.29	97	277.29	Y	N	0.92
EN24	5	242	147.06	89	169.08	N	N	0.69
EN30	2	95	189.87	33	322.09	N	N	0.8

Table 2. Performances of PAN 2013 Challenge for English Dataset

Submission	**English**		
	F_1	Precision	Recall
zhenshi13	0.800	0.800	0.800
seidman13	0.800	0.800	0.800
layton13	0.767	0.767	0.767
moreau13	0.767	0.767	0.767
jankowska13	0.733	0.733	0.733
ayala13	0.733	0.733	0.733
halvani13	0.700	0.700	0.700
feng13	0.700	0.700	0.700
ghaeini13	0.691	0.760	0.633
petmanson13	0.667	0.667	0.667
bobicev13	0.644	0.655	0.633
sorin13	0.633	0.633	0.633
vandam13	0.600	0.600	0.600
jayapal13	0.600	0.600	0.600
kern13	0.533	0.533	0.533
baseline	0.500	0.500	0.500
gillam13	0.500	0.500	0.500
vladimir13	0.467	0.467	0.467
grozea13	0.400	0.400	0.400

as the base one or not). The underlying idea is that the model describes a set of ways in which an author composes the sentences in its writings. If we can bring back such writing habits from the target model to the base model, we can conclude that the author is the same.

It must be underlined a small number of documents is sufficient, using this approach, to build an author's model. This is important because, in real life, only a few documents are available for the base author, on which basing a classification. We wanted to stress specifically this aspect in our experiments, using the training set released for the PAN 2013 challenge. Preliminary results are promising. Our approach seems practicable in real contexts since it does not need of a training phase and performs well also with short texts.

The current work in progress concerns the evaluation of our system using the original test set used in PAN 2013 challenge. As a future work, we plan to study the quality of the clusters, pursuing an intensional understanding thereof. In particular, we want to study whether generalizing the clustered clauses we can obtain a theory expressing the typical sentence construction that the author exploits in his texts. Such theory would be the intensional model of the author, which would allow to carry on the investigation in the learning field.

References

[1] Shlomo Argamon, Marin Saric, and Sterling Stuart Stein. Style mining of electronic messages for multiple authorship discrimination: first results. In Lise Getoor, Ted E. Senator, Pedro Domingos, and Christos Faloutsos, editors, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480. ACM, 2003.

[2] Shlomo Argamon, Casey Whitelaw, Paul Chase, Sobhan Raj Hota, Navendu Garg, and Shlomo Levitan. Stylistic text classification using functional lexical features: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 58(6):802–822, April 2007.

[3] Marie catherine De Marneffe, Bill Maccartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *In Proc. Intl Conf. on Language Resources and Evaluation (LREC)*, pages 449–454, 2006.

[4] Joachim Diederich, Jörg Kindermann, Edda Leopold, and Gerhard Paass. Authorship attribution with support vector machines. *Applied Intelligence*, 19(1-2):109–123, May 2003.

[5] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[6] Stefano Ferilli, Teresa M.A. Basile, Nicola Di Mauro, and Floriana Esposito. Plugging numeric similarity in first-order logic horn clauses comparison. In Roberto Pirrone and Filippo Sorbello, editors, *XIIth International Conference of the Italian Association for Artificial Intelligence*, volume 6934 of *LNCS*, pages 33–44. Springer, 2011.

[7] Stefano Ferilli, Fabio Leuzzi, and Fulvio Rotella. Cooperating techniques for extracting conceptual taxonomies from text. In *Proceedings of The Workshop on Mining Complex Patterns at AI*IA XIIth Conference*, 2011.

[8] D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2003.

[9] Fabio Leuzzi, Stefano Ferilli, and Fulvio Rotella. ConNeKTion: A tool for handling conceptual graphs automatically extracted from text. In Tiziana Catarci, Nicola Ferro, and Antonella Poggi, editors, *Bridging between Cultural Heritage Institutions – Proceedings of the 9th Italian Research Conference on Digital Libraries (IRCDL 2013)*, volume 385 of *CCIS*. Springer-Verlag Berlin Heidelberg, 2013.

[10] Fabio Leuzzi, Stefano Ferilli, and Fulvio Rotella. Improving robustness and flexibility of concept taxonomy learning from text. In Annalisa Appice, Michelangelo Ceci, Corrado Loglisci, Giuseppe Manco, Elio Masciari, and Zbigniew W. Ras, editors, *New Frontiers in Mining Complex Patterns - First International Workshop, NFMCP 2012, Held in Conjunction with ECML/PKDD 2012, Bristol, UK,*

September 24, 2012, Revised Selected Papers, volume 7765 of *CCIS*, pages 232–244. Springer-Verlag Berlin Heidelberg, April 2013.

[11] Jiexun Li, Rong Zheng, and Hsinchun Chen. From fingerprint to writeprint. *Commun. ACM*, 49(4):76–82, April 2006.

[12] David Lowe and Robert Matthews. Shakespeare vs. fletcher: A stylometric analysis by radial basis functions. *Computers and the Humanities*, 29(6):449–461, 1995.

[13] Philip M. Mccarthy, Gwyneth A. Lewis, David F. Dufty, and Danielle S. Mcnamara. Analyzing writing styles with coh-metrix. In Geoff Sutcliffe and Randy Goebel, editors, *In Proceedings of the Florida Artificial Intelligence Research Society International Conference (FLAIRS*, pages 764–769. AAAI Press, 2006.

[14] Long Qiu, Min-Yen Kan, and Tat-Seng Chua. A public reference implementation of the RAP anaphora resolution algorithm. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*, pages 291–294. European Language Resources Association, 2004.

[15] Sindhu Raghavan, Adriana Kovashka, and Raymond Mooney. Authorship attribution using probabilistic context-free grammars. In *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10, pages 38–42, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[16] Fulvio Rotella, Stefano Ferilli, and Fabio Leuzzi. An approach to automated learning of conceptual graphs from text. In Moonis Ali, Tibor Bosse, Koen V. Hindriks, Mark Hoogendoorn, Catholijn M. Jonker, and Jan Treur, editors, *Recent Trends in Applied Artificial Intelligence, 26th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2013, Amsterdam, The Netherlands, June 17-21, 2013. Proceedings*, volume 7906 of *Lecture Notes in Computer Science*, pages 341–350. Springer, 2013.

[17] Fulvio Rotella, Stefano Ferilli, and Fabio Leuzzi. A domain based approach to information retrieval in digital libraries. In Maristella Agosti, Floriana Esposito, Stefano Ferilli, and Nicola Ferro, editors, *Digital Libraries and Archives - 8th Italian Research Conference, IRCDL 2012, Bari, Italy, February 9-10, 2012, Revised Selected Papers*, volume 354 of *CCIS*, pages 129–140. Springer-Verlag Berlin Heidelberg, January 2013.

[18] Fiona J. Tweedie, S. Singh, and David I. Holmes. Neural network applications in stylometry: The federalist papers. *Computers and the Humanities*, 30(1):1–10, 1996.

[19] Hans van Halteren. Linguistic profiling for author recognition and verification. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[20] Rong Zheng, Jiexun Li, Hsinchun Chen, and Zan Huang. A framework for authorship identification of online messages: Writing-style features and classification techniques. *J. Am. Soc. Inf. Sci. Technol.*, 57(3):378–393, February 2006.

Thresholding of Semantic Similarity Networks using a Spectral Graph Based Technique

Pietro Hiram Guzzi, Simone Truglia, Pierangelo Veltri, and Mario Cannataro[1]

Department of Surgical and Medical Sciences, University Magna Graecia of Catanzaro
hguzzi@unicz.it

Abstract. Semantic similarity measures (SSMs) refer to a set of algorithms used to quantify the similarity of two or more terms belonging to the same ontology. Ontology terms may be associated to concepts, for instance in computational biology gene and proteins are associated with terms of biological ontologies. Thus, SSMs may be used to quantify the similarity of genes and proteins starting from the comparison of the associated annotations. SSMs have been recently used to compare genes and proteins even on a system level scale. More recently some works have focused on the building and analysis of Semantic Similarity Networks (SSNs) i.e. weighted networks in which nodes represents genes or proteins while weighted edges represent the semantic similarity score among them. SSNs are quasi-complete networks, thus their analysis presents different challenges that should be addressed. For instance, the need for the introduction of reliable thresholds for the elimination of meaningless edges arises. Nevertheless, the use of global thresholding methods may produce the elimination of meaningful nodes, while the use of local thresholds may introduce biases. For these aims, we introduce a novel technique, based on spectral graph considerations and on a mixed global-local focus. The effectiveness of our technique is demonstrated by using markov clustering for the extraction of biological modules. We applied clustering to simplified networks demonstrating a considerable improvements with respect to the original ones.

Keywords: Graphs, Semantic Similarity Measures, Thresholding

1 Introduction

The accumulation of raw experimental data about genes and proteins has been accompanied by the accumulation of functional information, i.e. knowledge about function. The assembly, organization and analysis of this data has given a considerable impulse to research [1]. Usually biological knowledge is encoded by using annotation terms, i.e. terms describing for instance function or localization of genes and proteins. Such annotations are often organized into ontologies, that offer a formal framework to organize in a formal way biological knowledge [2]. For instance, Gene Ontology (GO) provides a set of annotations (namely GO Terms) of biological aspects, structured into three main taxonomies: Molecular function

(MF), Biological Process (BP), and Cellular Component (CC). Annotations are often stored in publicly available databases, for instance a main resource for GO annotations is the Gene Ontology Annotation (GOA) database [3].

A set of algorithms, referred to as Semantic Similarity measures (SSMs), enabled the comparison of set of terms belonging to the same ontology. SSMs take in input two or more ontology terms and produce as output a value representing their similarity. This enabled the possibility to use such formal instruments for the comparison and analysis of proteins and genes [2].

Consequently, many works have focused on: (i) the definition of ad-hoc semantic measures tailored to the characteristics of Gene Ontology ; (ii) the definition of measures of comparison among genes and proteins; (iii) the introduction of methodologies for the systematic analysis of metabolic networks; (iv) building of *semantic similarity networks*, i.e. edge-weighted graph whose nodes are genes or proteins, and edges represent semantic similarities among them [4].

A semantic similarity network of proteins (SSN) is an edge-weighted graph G_{ssu}=(V,E), where V is the set of proteins, and E is the set of edges, each edge has an associated weight that represent the semantic similarity among related pairs of nodes.

These networks are constructed by computing some similarity value between genes or proteins. Nevertheless, such networks are usually quasi complete networks, so the use of them as framework of analysis has many problems.

Thus the definition of a threshold on the edge weight to retain only the meaningful relationships is a crucial step. An high threshold may result on the loss of many significant relationship while a low threshold may introduce a lot of noise-

In other kind of networks many methods have been defined: for instance the use of an arbitrary global threshold [5], or the use only of a fraction of highest relationship [6], or statistical based methods [7]. Nevertheless, internal characteristics of SSMs (as investigated in [8]) do not suggest the use of global thresholds. In fact small regions of relatively low similarities may be due to the characteristics of measures while proteins or genes have high similarity. Thus the use of local threshold may constitute an efficient way, i.e retaining only top k-edges for each node [9]. Although this consideration, this choice may be influenced by the presence of local noise and in general may cause the presence of biases in different regions.

Starting from these considerations, we developed a novel hybrid method that merges together both local and global considerations. This method is based on spectral graph theory and it is based on two main considerations.

We apply a local threshold for each node, i.e we retain only edges whose weight is higher than the average of all its adjacent. The choice of the threshold is made by considering a global consideration: the emergence of nearly-disconnected components by looking at the laplacian of the graph and its eigenvalues [10, 11]. In particular we build a novel graph in which edge weights are 0,5 and 1. The weight 0,5 is associated to edges that are retained considering only one adjacent node, while the weight 1 is associated to edges that are retained.

The choice of this simplification has a biological counterpart on the structure of biological networks. It has been proved in many works that these biological networks tend to have a modular structure in which hubs proteins (i.e. relevant proteins) have many connections [12–14]. Moreover, many works proved the existence of community structures, i.e. small dense regions with few link to other regions [15]. These considerations have usually inspired many algorithms for extracting biological relevant modules by analyzing biological networks [16].

From these consideration arises the main hypothesis of this paper: the simplification of quasi complete SSN by removing non relevant edges to evidence the formation of a structure of networks characterized by relatively-small dense networks loosely coupled with other ones.

After the application of the proposed simplification, we analyze resulting networks by applying a common algorithms used to mine graphs. We show that thresholded networks have in general more performances and that the best ones are reached with nearly-disconnected ones.

2 Problem Statement

We here introduce main concepts used for the formulation of the main problem of this article.

2.1 Spectral Graph Analysis

Spectral graph theory [17] refers to the study of the properties of a graph by looking at the properties of the eigenvalues and eigenvectors of matrices associated to the graph. In particular we here focus on the Laplacian matrix of a graph that is defined as follows [18, 19].

Given an edge-weighted graph G with n nodes, we may define the weighted adjacency matrix A as the nxn matrix in which the element $a_{i,j}$ is defined as follows.

$$a_{i,j} = \begin{cases} w_{i,j} \text{ if i,j are connected;} \\ 0, \quad \text{if i,j are not connected} \end{cases} \tag{1}$$

For these graphs the notion of degree may be easily extended in this way. For each vertex v_i the degree is defined as the sum of the weights of all the adjacent edges $vol_{v_i} = \Sigma_j w_{i,j}$. Then we may define the Degree Matrix D as follows:

$$d_{i,j} = \begin{cases} vol_{v_i}, \text{ if i=j;} \\ 0, \quad \text{elsewhere} \end{cases} \tag{2}$$

Finally, the Laplacian Matrix L is defined as $L = D - A$. Similarly in literature other slightly definitions of Laplacian (e.g. Signless Laplacian, Normalized Laplacian [20]) have been proposed.

Beside the other properties that are related to the characteristic polynomial of laplacian, we here focus on the smallest nonzero eigenvalue, often referred to as Fiedler vector [21]. It has been shown that the number of connected components

is related to the algebraic multiplicity of the smallest eigenvalues in case of both un-weighted and weighted graphs. Starting from this consideration, Ding et al. [10] observed that also nearly-disconnected components may also identified by analyzing the eigenvector associated to the Fiedler vector.

For this study we analysed the spectrum of the graph obtained after the simplification under the hypothesis that a graph with nearly disconnected component may represent a suitable choice. If the graph is connected we will build a novel graph. If the graph has-nearly disconnected component we end the process and we mine the resulting subgraph for the identification of biological relevant modules.

2.2 Semantic Similarity Measures

A semantic similarity measure ($SSMs$) is a formal instrument to quantify the similarity of two or more terms of the same ontology. Measures comparing only two terms are often referred to as pairwise semantic measures, while measures that compare two sets of term yielding a global similarity among sets are referred to as groupwise measures.

Since proteins and genes are associated to a set of terms coming from Gene Ontology, $SSMs$ are often extended to proteins and genes. Similarity of proteins is then translated in the determination of similarity of set of associated terms [22, 23]. Many similarity measures have been proposed (see for instance [2] for a complete review) that may be categorized according to different strategies used for evaluating similarity. We here do not discuss deeply $SSMs$ for lack of space.

3 The Proposed Approach

We here introduce a method for threshold selection on weighted graph based on the spectrum of the associated laplacian matrix. The process is straightforward. The pruning algorithm examines each node in the input graph. For each node it stores all the weights of the adjacent edges. Then it determines a local threshold $k = \mu + \alpha * sd$, where μ is the average of weights, sd is the standard deviation and α is a variable threshold that is fixed globally. In this way we realize an hybrid approach since the threshold k has a global component α and a local one given by the average and standard deviation of the weights of the adjacent.

If the weight of an edge is greater than k considering the adjacent of both its nodes, then it will be inserted into the novel graph with unitary weight. Otherwise, hen if the weight of an edge is greater than k considering only one of its adjacent nodes, then it will be inserted into the novel graph with weight 0,5. At the end of this process, the Laplacian of the spectrum of the graph is analyzed as described in Ding et al [10]. If the graph presents nearly disconnected components, then the process stops, alternatively a novel graph with a more stringent threshold k is generated.

3.1 Building Semantic Similarity Networks

Following algorithm explains the building of the semantic similarity network G_{ssu} by iteratively calculating semantic similarity among each pair of proteins. For each step two proteins are chosen and the semantic similarity among them is calculated. Then nodes are added to the graph and an edge is inserted when the semantic similarity is greater than 0.

Algorithm 1: Building Semantic Similarity Networks

```
Building Semantic Similarity Networks Data: Protein Dataset P,
        Semantic Similairity Measure SS
Result: Semantic Similarity Network G_ssu=V_ssu, E_ssu
initialization;
forall the p_i in P do
    read p_i;
    add p_i in V_ssu ;
    forall the p_j in P, j ≠ i do
        Let σ=SS(p_i,p_j) ;
        if alpha is greater than 0 then
            add the weighted edge (p_i,p_j,σ) to E_ssu;
        end
    end
end
```

3.2 Pruning Semantic Similarity Networks

This section explains the pruning of semantic similarity network through an example. To better clarify the process, we use an auxiliary graph G_{pr} that is the final process of pruning. The graph is built in an incremental fashion by considering all the nodes of G_{ssu}. The process is straightforward. The pruning algorithm examines each node $\in G_{ssu}$. For each node it stores all the weights of the adjacent edges. Then it determine a local threshold (for instance the average of the weights or the median value as exposed after). At the end of this step, the node i and all the adjacent ones are inserted in to G_{pr} (only if they are not yet present).

Then each edge adjacent to i with weight greater with the determined local threshold is inserted into G_{pr}. If the considered edge is not present in G_{pr}, the edge will have weight 0.5, otherwise the weight of the edge is set to 1. We used in this work two simple thresholds, the average and the median of all the weights. Finally all the nodes with 0 degree are deleted from G_{pr}.

The rationale of this process is that edges that are *relevant* considering the neighborhood of both nodes will compare in the pruned graph with unitary

weight while edges that are *relevant* considering one node will compare with 0.5 weight. In this way we think that we may reduce the noise.

For instance, let us consider the network depicted in Figure 1 and let us suppose that threshold is represented by the average. Without loss of generality we suppose k=0 in this example. Let $AVG(node_i)$ be the average of the weights of nodes adjacent to $node_i$ that is used as threshold.

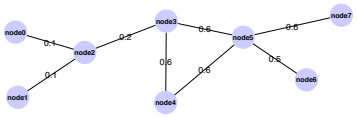


Fig. 1. Weighted Semantic Similarity Network.

- The algorithm initially explores $node_0$, since it has degree 1, it is discarded from the analysis.
- Then it explores $node_1$ that is discarded similarly to $node_0$.
- When $node_2$ is considered, the algorithm adds into G_{pr} $node_0, node_1, node_2$, and $node_4$ and the edge $(node_2, node_4)$ with weight 0.5 - (the average of the weights of the neighbours of $node_2$ is equal to 0,13 and other two edges have a lower weight). Figure 2 depicts the produced graph at this step.

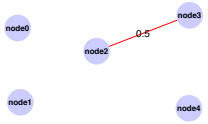


Fig. 2. The output of the algorithm at Step 2

- $node_3$ is reached by the visiting. Then $node_4$, and $node_5$ are inserted into G_{pr}. The $AVG(node_3)$ is equal to 0,46, so only edges $(node_3, node_5)$ and $(node_3, node_5)$ are inserted into G_{pr} with weight 0.5. Figure 3 depicts G_{pr} after this step.
- $node_4$ is reached. Since all the adjacent nodes have been inserted into G_{pr}, no nodes are added into this step. The $AVG(node_4)$ is equal to 0,6, so all the edges must be inserted. In particular edge $(node_4, node_3)$ is yet present, so its weight is updated to 1.0. Diversely, $(node_4, node_5)$ is inserted with weight equal to 0.5. Figure 4 depicts G_{pr} after this step.
- $node_5$ is reached. $node_7$ and $node_8$ are inserted into G_{pr}. The $AVG(node_5)$ is 0,575. Consequently the weight of $(node_5, node_3)$ $(node_5, node_3 4$ and in

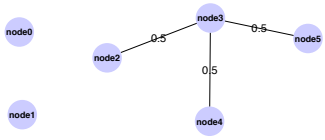


Fig. 3. Output after the visit of node3.

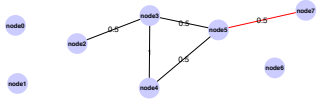


Fig. 4. Output after the visit of node4.

G_{pr} is updated to 1, $(node_6, node_7)$ is inserted into G_{pr}.Figure 6 depicts G_{pr} after this step.

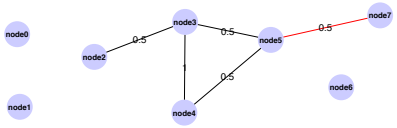


Fig. 5. Output after the visit of node5.

- $node_7$ and $node_8$ are visited but discarded since they have degree equal to 1.
- Finally all the nodes with zero degree are eliminated from G_{pr}, producing the resulting graph depicted in Figure 6.

The generation of pruned graph is repeated until the graph has nearly disconnected components. This may be evident by analyzing the spectrum of the associated laplacian for value of threshold.

```
Pruning Semantic Similarity Network
 Input SSn Raw Semantic Simlarity Network,
 K  Threshold of Simplification
 Output: SSp Simplified Semantic Similarity Network
  While SSp has not nearly-disconnected component
  $SSp$ = Simplify(SSn,k)
   Increment k
Return: SSp
```

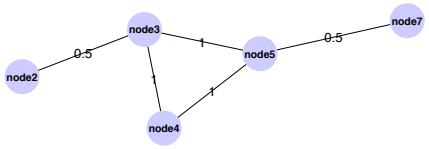


Fig. 6. Final pruned graph

3.3 Analysis of Semantic Similarity Networks

As introduced, in a Semantic Similarity Networks, nodes represent proteins or genes, and edges represent the value of similarity among them. Starting from a dataset of genes or proteins, a SSN may be built in an iterative way, and once built, algorithms from graph theory may be used to extract topological properties that encode biological knowledge.

As starting point, the global topology of an semantic similarity network, i.e. the study of the clustering coefficient or of the diameter, can reveal main properties of the network and the correspondence with respect to a theoretical model.

In addition to analysis of global properties, the study of recurring local topological features and the extraction of relevant modules, i.e. cliques, has found an increasing interest. For the purposes of this work, we focus on the extraction of dense subgraphs under the hypothesis that they could encode protein complexes.

SS measures are able to quantify the functional similarity of pairs of proteins/genes, comparing the GO terms that annotate them. Thus, there are no constraints on the minimum set size [2].

Since proteins within the same pathway are involved in the same biological process, they are likely to have high semantic similarity. In a similar way, protein belonging to the same complex are likely to have similar biological roles, and therefore they should have high semantic similarity.

The rationale of this study is to demonstrate the ability of semantic similarity networks to represent in a similar way to protein interaction networks. Main difference is represented by the fact that semantic similarity networks may encode more knowledge that is hidden in protein interaction networks.

There exist currently main approaches of analysis of protein interaction networks that span a broad range, from the analysis of a single network by clustering to the comparison of two or more networks trough graph alignment approaches. In this work we consider the use of Markov Clustering Algorithm (MCL) as mining strategy. MCL has been proved to be a good predictor of functional modules when applied to protein interaction networks.

4 Case Study

In order to show the effectiveness of this strategy we propose the following assessment:

– we downloaded three dataset of proteins (the CYC2008 dataset [1], the MIPS catalog [24], and the Annotated Yeast High-Throughput Complexes [2]);
– we calculated different semantic similarities among them using FastSemSim tool [3] (we considered 10 semantic similarity measures from those available in FastSemSim (Czekanowsky-Dice , Dice, G-Sesame, Jaccard, Kin, NTO, SimGic, SimICND, SimIC, SimUI, TO [25]) and two ontologies Biological Process (BP) and Molecular Function (MF). Consequently we generated 20 SSN for each input dataset.
– we applied the pruning of the semantic network with varying threshold causing the presence of nearly disconnected components and the presence of disconnected components;
– we extracted modules on the raw and simplified networks at various threshold showing the improvements of our strategy showing the improvement in terms of functional enrichment of modules (i.e. the quantification of biological meaning of modules).

As final step we compare our simplification with other global strategies demonstrating the effectiveness of the local simplification.

4.1 Results

For each generated network we used the markov clustering algorithm (MCL) to extract modules. The effectiveness of the use of MCL for detecting modules in networks has been demonstrated in many works (see for instance [25]). We here assess how MCL is able to discover *functionally coherent* modules in different semantic similarity network and how this process is positively influenced by the simplification. In particular we show how the process of simplification improves the overall results and how best results are obtained when networks presents nearly disconnected components.

We evaluated the obtained results in terms of *functional coherence* of extracted modules. We define *functional coherence* FC of a module M as the average of semantic similarity values of all the pair of nodes (i,j) composing a module.

$$\sum_{i,j} \frac{SSM(i,j)}{N}$$

, where N is the number of the proteins of the module.

Starting from this definition, we may obtain a single value for all the modules extracted in an execution of MCL by averaging these values. We consider this average value as a representative for the thresholded network. Figures 7, 8, and 9 summarize these results.

[1] wodaklab.org/cyc2008/
[2] wodaklab.org/cyc2008/
[3] fastsemsim.sourceforge.net

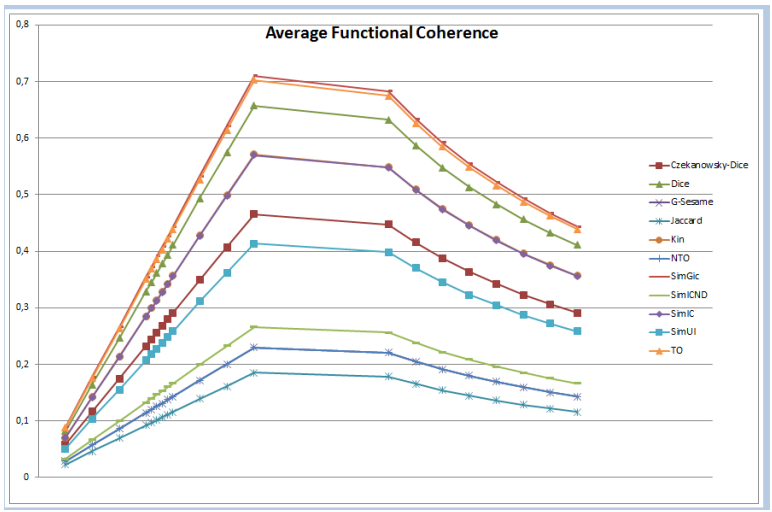


Fig. 7. Comparison of Average FC at different Threshold Levels on CYC2008 Dataset

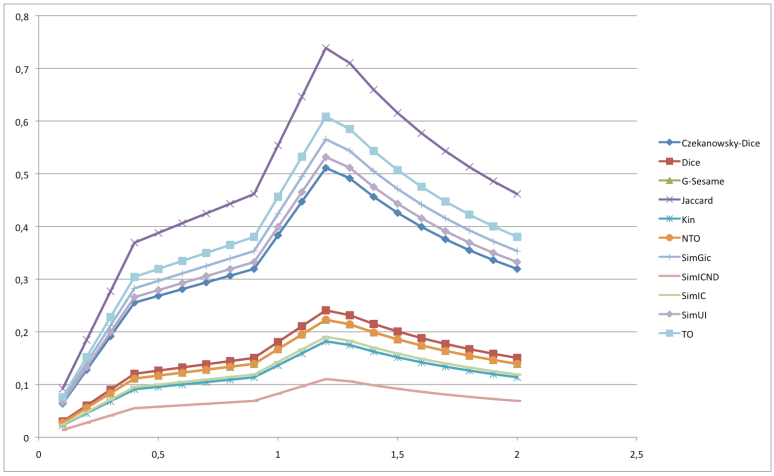


Fig. 8. Comparison of Average FC at different Threshold Levels on MIPS

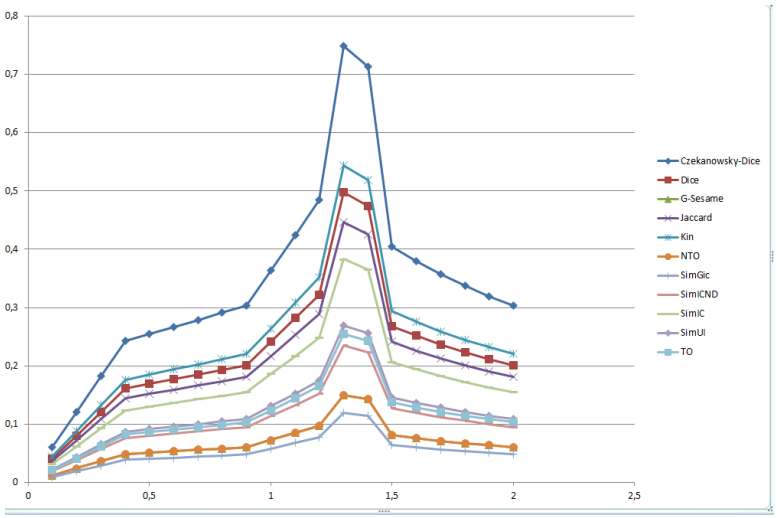


Fig. 9. Comparison of Average FC at different Threshold Levels on Annotated High Throughput Complexes Dataset

5 Conclusion

Results showed that raw semantic similarity networks contains lot of noise, thus are unsuitable for the analysis. Consequently we proposed a local simplification of networks. Result confirm that mining of simplified networks is a suitable way for extract biologically meaningful knowledge.

References

1. Cannataro, M., Guzzi, P.H., Sarica, A.: Data mining and life sciences applications on the grid. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **3**(3) (2013) 216–238
2. Guzzi, P., Mina, M., Guerra, C., Cannataro, M.: Semantic similarity analysis of protein data: assessment with biological features and issues. Briefings in bioinformatics **13**(5) (2012) 569–585
3. Camon, E., Magrane, M., Barrell, D., Lee, V., Dimmer, E., Maslen, J., Binns, D., Harte, N., Lopez, R., Apweiler, R.: The gene ontology annotation (goa) database: sharing knowledge in uniprot with gene ontology. Nucl. Acids Res. **32**(suppl_1) (January 2004) D262–266
4. Pesquita, C., Faria, D., Falcão, A.O., Lord, P., Couto, F.M.: Semantic similarity in biomedical ontologies. PLoS computational biology **5**(7) (July 2009) e1000443
5. Freeman, T., Goldovsky, L., Brosch, M., van Dongen, S., Maziere, P., Grocock, R., Freilich, S., Thornton, J., Enright, A.: Construction, visualization, and clustering of transcription networks from microarray expression data. PLoS Computational Biology **3**(10) (2007) e206

6. Ala, U., Piro, R., Grassi, E., Damasco, C., Silengo, L., Oti, M., Provero, P., Cunto, F.: Prediction of human disease genes by human-mouse conserved coexpression analysis. PLoS Computational Biology **4**(3) (2008) e1000043
7. Rito, T., Wang, Z., Deane, C.M., Reinert, G.: How threshold behaviour affects the use of subgraphs for network comparison. Bioinformatics **26**(18) (2010) i611–i617
8. P., G., M, M.: Investigating bias in semantic similarity measures for analysis of protein interactions. In: Proceedings of 1st International Workshop on Pattern Recognition in Proteomics, Structural Biology and Bioinformatics (PR PS BB 2011). (13th September 2011 2012) 71–80
9. Lee, H., Hsu, A., Sajdak, J., Qin, J., Pavlidis, P.: Coexpression analysis of human genes across many microarray data sets. Genome Res **14** (2004) 1085–1094
10. Ding, C., He, X., Zha, H.: A spectral method to separate disconnected and nearly-disconnected web graph components. Proceedings of the Seventh ACM International Conference on Knowledge Discovery and Data Mining: 26-29 August 2001; San Francisco (2001)
11. Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: analysis and an algorithm. Advances in Neural and Information Processing Systems: 3-8 December 2001; Vancouver (2001)
12. Ma, X., Gao, L.: Biological network analysis:insights into structure and functions. Briefings in Functional Genomics **11**(6) (2012) 434–442
13. : On the functional and structural characterization of hubs in protein-protein interaction networks. Biotechnology Advances **31**(2) (2013) 274 – 286
14. Zhu, X., Gerstein, M., Snyder, M.: Getting connected: analysis and principles of biological networks. Genes & Development **21**(9) (2007) 1010–1024
15. Su, G., Kuchinsky, A., Morris, J.H., States, D.J., Meng, F.: Glay: community structure analysis of biological networks. Bioinformatics **26**(24) (2010) 3135–3137
16. Ji, J., Zhang, A., Liu, C., Quan, X., Liu, Z.: Survey: Functional module detection from protein-protein interaction networks. IEEE Transactions on Knowledge and Data Engineering **99**(PrePrints) (2013) 1
17. Chung, F.: Spectral graph theory. Regional Conference Series in Mathematics, Providence: American Mathematical Society **92** (1994)
18. Cvetković, D., Simić, S.K.: Towards a spectral theory of graphs based on the signless laplacian, ii. Linear Algebra and its Applications **432**(9) (2010) 2257–2272
19. : Spectra and optimal partitions of weighted graphs. Discrete Mathematics **128**(13) (1994) 1 – 20
20. Merris, R.: Laplacian matrices of graphs: a survey. Linear algebra and its applications **197** (1994) 143–176
21. Mohar, B.: The laplacian spectrum of graphs. In: Graph Theory, Combinatorics, and Applications. Volume 2. (1991) 871–898
22. Pesquita, C., Faria, D., Falcao, A., Lord, P., Couto, F.M.: Semantic similarity in biomedical ontologies. PLoS Comput Biol **5**(7) (07 2009) e1000443
23. Wang, H., Zheng, H., Azuaje, F.: Ontology- and graph-based similarity assessment in biological networks. Bioinformatics **26**(20) (October 2010) 2643–2644
24. Guldener, U., Munsterkotter, M., Oesterheld, M., Pagel, P., Ruepp, A., Mewes, H., Stumpflen, V.: Mpact: the mips protein interaction resource on yeast. Nucleic Acids Res **34** (2006) D436–441
25. Cannataro, M., Guzzi, P.H., Veltri, P.: Protein-to-protein interactions: Technologies, databases, and algorithms. ACM Comput. Surv. **43** (December 2010) 1:1–1:36

A Study on Parameter Estimation for a Mining Flock Algorithm

Rebecca Ong[1], Mirco Nanni[1], Chiara Renso[1], Monica Wachowicz[3], and Dino Pedreschi[2]

[1] KDDLab, ISTI-CNR, Pisa,Italy
[2] KDDLab, University of Pisa, Italy
[3] University of New Brunswick, Fredericton, Canada

Abstract. Due to the diffusion of location-aware devices and location-based services, it is now possible to analyse the digital trajectories of human mobility through the use of mining algorithms. However, in most cases, these algorithms come with little support for the analyst to actually use them in real world applications. In particular, means for understanding how to choose the proper parameters are missing. This work improves the state-of-the-art of mobility data analysis by providing an experimental study on the use of data-driven parameter estimation measures for mining flock patterns. Experiments were conducted on two real world datasets, one dealing with pedestrian movements in a recreational park and the other with car movements in a coastal area. The study has shown promising results for estimating suitable values for parameters for flock patterns envisaging a formal framework for parameter evaluation in the near future, since the advent of more complex pattern algorithms will require the use of a larger number of parameters.

1 Introduction

The increasing availability of data pertaining to the movements of people and vehicles, such as GPS trajectories and mobile phone call records, has fostered in recent years a large body of research on the analysis and mining of these data, to the purpose of discovering the patterns and models of human mobility. Examples along this line include [2, 3], which highlight the broad diversity of mobility patterns. A few authors concentrated on the problem of characterising and detecting *flocks*, i.e., patterns describing a set of objects that stay closely together during an interval of time, either moving together along the same route (a *moving flock*), or staying together in a specific location (a *stationary flock*) [6, 4, 7].

Flocking patterns highlight groups with synchronised movements that stay together for a while and disappear afterwards. In this paper, we follow the definition where a flock is a group of at least k objects that, observed during a time interval ΔT with a sampling rate R, remain spatially close to each other within a distance ϵ. While this definition and other variations in literature are useful for detecting flocks, it is apparent that setting such parameters $(k, \epsilon, \Delta T, R)$ makes

it complex for an analyst to use flock mining in different contexts. These parameters clearly depend on the data under analysis, and may be vary greatly in different settings. We observed remarkable differences in datasets pertaining to pedestrian and car movements, which are the two trajectory datasets used in this paper. Such differences can be expected as well when observing other types of moving objects, e.g., bird trajectories. Despite this complexity, no prior work addressed the problem of parameter setting, which is a barrier especially for mobility experts who would like to use flock mining as a black box. To this aim, we address the *parameter estimation* problem of finding appropriate values for the parameters using a systematic data-driven method, based on the trajectory dataset that is being analysed. This paper provides an empirical evaluation of the effects of parameters in two different moving objects datasets. This is an initial step towards delineating a data-driven parameter estimation method for flock mining.

The structure of this paper is as follows: Section 2 presents some related approaches in the literature. Section 3 provides a summary of the flock algorithm considered in this study while Section 4 describes the experiments performed on two datasets in order to study the effect of the different parameters. Finally, Section 5 sums up the conclusions derived from the study.

2 Related works

Although the problem of finding realistic parameter values in data mining is well recognized in literature, very few papers have addressed this problem. Paper [1] is a well known work that proposes a solution for parameter estimation, and has inspired our approach. In this work, the authors propose a heuristic technique for determining the parameter values of the density-based clustering algorithm DBSCAN: Eps (radius) and $MinCard$ (minimum cardinality of clusters). A function called *k-distance* is defined to compute the distance between each object and its k-th nearest neighbor. These values are then plotted with objects ordered in descending order, in the so-called *sorted k-distance plot*. This plot is then used to find an estimation of the parameters: given an object p, the k-distance(p) is the value of Eps while $MinCard$ is set to $k+1$. This setting means that all objects with an equal or smaller k-distance are the "core" objects. The threshold object which maximize the k-distance in the least dense cluster gives the desired values. This object can be found visually in the plot by identifying the first "valley". The objects plotted to the left of the threshold will be considered as noise while other objects will be assigned to some cluster.

Another related work on parameter estimation related to flocks is found in [5] where the authors propose a set of algorithms for detecting convoys in trajectory datasets. They proposed a guideline for determining the parameters δ and λ of the proposed Douglas-Peucker (DP) algorithm, with the purpose of optimizing the execution time. The optimal convoy algorithm is run on a pre-processed dataset where trajectories have been simplified using the DP algorithm, which uses δ as a tolerance value for constraining the spatial distance between the

original and the simplified points. The algorithm uses another additional parameter λ, which refers to the length of the time partitions. The determination of a good value for δ has the goal of finding a trade-off value giving a good simplification of original trajectories while mantaining a tight enough distance. For finding a good value for δ authors propose to run the DP algorithm with δ set to 0. They consider the actual tolerance values at each simplification step and find the values with the largest difference with their neighbour before averaging them to obtain the final parameter value. Meanwhile, a good λ is computed by taking the average probability of each object having an intermediate simplified point that is not found in other trajectories. However, the parameter estimation techniques were applied to the preprocessing step of the convoy algorithm rather than applying it directly to the parameters related to the flock or convoy definition.

3 A Moving Flock Extraction Algorithm

The study for parameter estimation has been designed with reference to the flock algorithm introduced in [7]. The algorithm finds moving flocks, each of which is a group of objects consisting of at least *min_points* members that are spatially close together while moving from one location to another over a minimum time duration *min_time_slices*. The algorithm requires four user-defined parameters: *synchronisation_rate*(R) - refers to the rate, specified in seconds, at which observation points (e.g., GPS recordings) are sampled for each moving object; *min_time_slices*(ΔT) - is the minimum number of consecutive times slices for which the objects remain spatially close; *min_points*(κ) - is the minimum number of objects in a moving flock; *radius*(ϵ) - defines the spatial closeness of a group of moving objects at a specific time instance.

The following is a pseudocode of the moving flock algorithm:

Algorithm 1: Moving Flock($\mathcal{D}$, R, ΔT, κ, ϵ)

```
1 synchDataset = synchronise(D, R));
2 for each traj in synchDataset
3   if traj is NOT marked
4     for each curr_point, a sampled point of current traj
5       1-flocks = computeSpatialNeighbour(curr_point, ε, synchDataset);
6     n-flocks = merge_adj_cand(1-flocks);
7     for each cand, a candidate flock in n-flocks
8       if count_time_slices(cand)>=ΔT and compute_extent(cand)>=ε
9         for each traj', a member of cand
10            mark traj';
11        F = F ∪ cand;
```

The algorithm initially samples the observation points in the input dataset at a regular interval R (line 1). The next steps are performed for each of the

trajectories. The current unmarked trajectory is considered as the base member and for each of its time instances, points belonging to other trajectories are considered as neighbours if their x,y components are within ϵ distance to that of the base (lines 4-5), producing candidate flocks lasting for only 1 time slice (i.e., $1{-}flocks$). The $1{-}flocks$ with adjacent time slices and have at least κ members in common are then merged, producing $2{-}flocks$ candidates. Merging is recursively applied until $n{-}flocks$, which refer to the longest duration candidate flocks, are found (line 6). $n{-}flocks$ lasting for at least ΔT time slices and covering a spatial extent of at least ϵ are considered as moving flocks and their member trajectories are marked in order to reduce the number of base trajectories that need to be processed (lines 8-11).

4 Parameter Estimation

This section focuses on the investigation of the effects of individual parameters on the flock results to understand how suitable values can be selected. This study for parameter estimation has been designed with reference to the flock algorithm introduced in [7], to find moving flock patterns using the parameters k, ϵ, ΔT, and R as discussed in section 3.

We start with a description of the datasets used for the experiments and an overview of some flock quality measures since these are necessary to understand the impact of the parameters on the results. These are followed by a discussion of the effect of each parameter before closing with an approach on finding a suitable *radius* value.

4.1 Context Awareness and Flock Cohesion Distance

We performed the study on two datasets that have two entirely different settings of two different types of moving objects. The first dataset, called DNP, contains 370 trajectories, one for each visitor and consisting of a total of 141,826 sample points. These were recorded using GPS devices given to the visitors at the parking lots where they have started their visits. Due to the sparsity of this dataset, we have combined the data in different days into one day. The second dataset, called OctoPisa, contains the trajectories of $\approx$ 40,000 cars for a total of $\approx$ 1,500,000 travels covering a temporal span of 5 weeks in a coastal area of Tuscany around the city of Pisa. From this large dataset, we concentrated on a subset of trajectories occurring on June, 29, 2010 in order to be able to perform a more detailed study on a specific time period. This is one of the days with the highest number of moving cars. It contains 28,032 trajectories (corresponding to 557,201 observed GPS points) of 10,566 vehicles. The flock algorithm can be applied as well to the other days and the obtained flocks can be combined in a straightforward manner to obtain the flocks inherent in the entire dataset.

The initial set of parameter values that we used for the DNP dataset is as follows: $R = 5mins.$, $\Delta T = 3$, $\kappa = 3$, and $\epsilon = 150m$. Meanwhile, the initial set used for the OctoPisa dataset is as follows: $R = 1min.$, $\Delta T = 3$, $\kappa = 3$,

and $\epsilon = 150m$. For both datasets, we maximized R to a value that does not cause large distortion (from the domain expert's perspective) among the input trajectories. We selected a value of 3 for both ΔT and κ since using 2 is too small while 4 is quite large for finding a good number of flocks. Then, using the values for R, ΔT, κ, and the type of entity (i.e., pedestrian and car) in consideration, we derived a feasible and logical value for ϵ. In observing the effects of the individual parameters, we only modify the value of the parameter in consideration and retain the initial values for the rest.

A first step in parameter estimation is to understand how the parameters influence the results obtained by the flock extraction algorithm. In doing so, it is important to have an objective measure of this influence in order to understand whether decreasing or increasing the parameter values improves or worsens the quality of discovered flocks.

In our study, we used three measures, which are extensions of measures used for cluster evaluation. These measures include cohesion, separation and silhouette coefficient.

Flock cohesion distance is a measure of spatial closeness among members of a discovered flock. It is analogous to the cohesion measure used for evaluating clusters but specifically applied to flock patterns. It can be computed using Equation 1, which evaluates a specific flock F_i by computing the distance between each flock member m_j with the base $m_{i(b)}$. Recall from Algorithm 1 that candidate flocks are found using each trajectory as a base. Each discovered flock F_i has $m_{i(b)}$ as its base member. Members of F_i are spatially close to $m_{i(b)}$ for its duration of flocking. $|F_i|$ is the number of F_i's members.

$$flock\ coh(F_i) = \frac{\sum_{\substack{m_j \in F_i \\ m_j \neq m_{i(b)}}} prox_{intra}(m_j, m_{i(b)})}{|F_i| - 1} \tag{1}$$

The $prox_{intra}$ between a flock member m_j and the flock base $m_{i(b)}$ can be computed by averaging the Euclidean distance among (x, y) points that were sampled simultaneously as described in Equation 2. T refers to the flocking duration and it consists of a set of sampled time instances. x_j^t and y_j^t refer to the x, y components of member j at time instance t. $x_{i(b)}^t$ and $y_{i(b)}^t$ are the x and y components of flock i's base.

$$prox_{intra}(m_j, m_{i(b)}) = \frac{\sum_{t \in T} euclDist((x_j^t, y_j^t), (x_{i(b)}^t, y_{i(b)}^t))}{|T|} \tag{2}$$

The overall flock cohesion distance of an obtained flock result can be computed by averaging the flock cohesion distance scores for each flock in the result as shown in Equation 3. Naturally, a flock with a low cohesion distance score is considered as a high quality flock.

$$overall\ flock\ cohesion\ distance(F) = \frac{\sum_{F_i \in F} flock\ coh(F_i)}{|F|} \tag{3}$$

On the contrary, flock separation is a measure of spatial or spatio-temporal detachment of a flock from the rest and it can be computed using Equation 4.

$$flock\ sep(F_i) = \sum_{\substack{F_j \in F \\ i \neq j}} prox_{inter}(m_{b(i)}, m_{b(j)}) \tag{4}$$

While $prox_{intra}$ measures the distance among members of a flock, $prox_{inter}$ measures the distance among different flocks. The distance between a pair of flocks is computed by computing the distance between their respective bases. We propose two approaches for this computation: $prox_{inter(XYT)}$ and $prox_{inter(routeSim)}$. $prox_{inter(XYT)}$ considers the spatio-temporal components in computing the distance while $prox_{inter(routeSim)}$ only considers the spatial components. Using $prox_{inter(routeSim)}$, distance is computed based on the similarity between the route followed by the flocks, without considering co-occurrence of the route similarity in time. Choosing between these two depends on the similarity level that the user is interested in.

$prox_{inter(XYT)}$ computes the spatial distance among the portion of the base trajectories that overlap in time as was done for $prox_{intra}$. The remaining portion that does not overlap incurs a penalty *pnlty*, which is the maximum possible distance obtained from the overlapping portion plus an arbitrary value. In the case that this maximum value does not exist since the bases being compared are disjoint, a maximum penalty score is incurred. More specifically, Equation 5 describes how $prox_{inter(XYT)}$ is computed. $nonOverlapLTI$ refers to the number of un-matched time instances in the longer trajectory, *euclDistOvlp* is the sum of Euclidean distances among pairs of points (from each base) that overlap in time, and $maxTD$ refers to the length of the longer base trajectory in terms of the number of time instances.

$$prox_{inter(XYT)} = \frac{pnlty * (nonOverLTI) + euclDistOvlp}{maxTD * (|F_i| - 1)} \tag{5}$$

For $prox_{inter(routeSim)}$, we adapted an existing algorithm for computing the route similarity distance. The algorithm ignores the temporal component of the bases and computes the distance in terms of the spatial components by comparing the shape of the trajectories.

As with the overall flock cohesion of an obtained flock result, the overall flock separation can be computed by averaging individual flock separation scores.

Finally, the flock silhouette coefficient is a combination of the previously discussed measures as shown in Equation 6. Note that computed scores can range from -1 (large intraflock distances and small interflock distances) to 1 (small intraflock distances and large interflock distances).

$$flock\ Sil(F_i) = \frac{flock\ sep(F_i) - flock\ coh(F_i)}{max\{flock\ sep(F_i), flock\ coh(F_i)\}} \tag{6}$$

As with overall flock cohesion and separation, the overall silhouette coefficient of a flock result can be computed by the averaging silhouette score of each flock.

4.2 Observing the Effect of Varying the Parameters

This part discusses the observations derived from investigating the effect of different parameter values on the obtained flock results for the DNP and the OctoPisa datasets. The following subsections provides a discussion of the individual effect of each parameter.

Effect of *synchronisation_rate* (R) Out of the 4 parameters of the algorithm, the *synchronisation_rate* can affect the quality of the input dataset. More specifically, a very large value of R can distort the input trajectories whereas a very small value requires a longer processing time.

We have observed how dataset cohesion changes for different values of R. Dataset cohesion describes how each individual trajectory is cohesive with respect to the rest of trajectories in the dataset. To compute this value, we applied the flock separation measure and treated each trajectory as a base trajectory.

Experiments demonstrate that the synchronisation step can indeed modify the input and its cohesiveness but at the same time, the variation is small in the two datasets for smaller values of R. Using XYT cohesion, the largest difference between the smallest cohesion score compared to the other scores obtained using larger R is 371.67m when $R = 11mins.$ in the DNP dataset. Meanwhile, the largest difference is 1987.77m using route similarity cohesion when $R = 15mins.$ in the DNP dataset. For the Octopisa dataset, the largest difference is 480.28m and 10789.69m using XYT and route similarity cohesion, respectively. The route similarity cohesion score varied more compared to the XYT cohesion score. This plot thus suggests the use of smaller values of R.

We now present the effect of the *synchronisation_rate* on the discovered flocks themselves. Figure 1 illustrates how different values of R can affect the flock results by observing the change in the number of moving flocks discovered, the overall flock cohesion, the overall flock separation (based on the spatio-temporal coordinates XYT and route similarity), and the overall silhouette coefficient (XYT and route similarity based).

Figure 1 shows the plots obtained for the *synchronisation_rate* in the two datasets. In general, fewer flocks are found as R increases. Furthermore, the overall flock cohesion varies slightly for different R values, while the overall flock separation tend to decrease with increasing R values. In the case of $R = 8mins.$ in Octopisa and $R = 13mins.$ in DNP, the discovered flocks becomes 0 and hence, the cohesion and separation scores are no longer applicable. Considering the plots for the XYT and route similarity separation scores, it is advisable to set R to a value less than $6mins.$ in DNP and a value less than $4mins.$ in Octopisa due to the sudden drop in the separation scores. A sudden drop occurs when no flock or only a single flock is discovered, or when the distance among the discovered flocks is small. Very large XYT separation scores indicates that the flocks are temporally disjoint. On the other hand, very large route similarity scores indicates that different flocks are following different routes. Finally, the silhouette coefficients summarize the effect of R on both the cohesion and separation scores. The silhouette coefficients are generally close to 1 (i.e., ideal

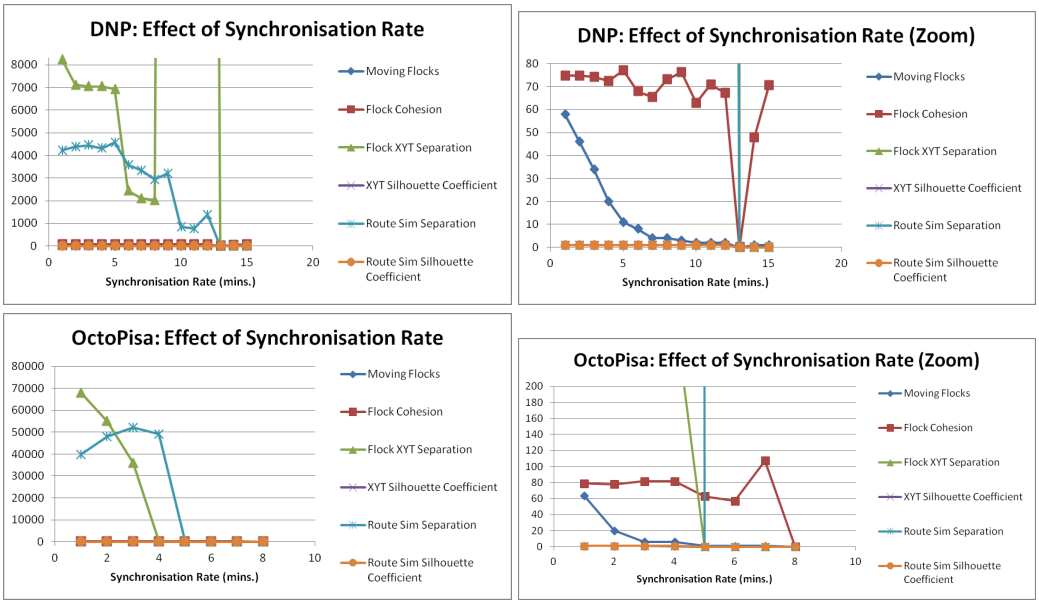


Fig. 1. Effect of the *synchronisation_rate* parameter for the two datasets. We have the full plots on the left part and the zoom in on the right part. The zoom in figures show the variation in the measures that have very small scores compared to the XYT and route similarity separation scores.

case), except for cases wherein the silhouette coefficient is 0. These cases refer to instances wherein only a single flock or no flock was found, making the silhouette coefficient inapplicable.

Effect of the *min_time_slices* (ΔT) Parameter *min_time_slices* and *synchronisation_rate* are parameters that are both related to time. Since the plots and observations for these parameters are generally similar, we no longer present the plots for *min_time_slices*.

As observed with *synchronisation_rate*, an increasing value of ΔT results in fewer number of discovered moving flocks, and, generally, lowering the XYT and route similarity separation scores. The XYT- and route similarity-based silhouette coefficients are either close to 1 when more than a single flock is found, or 0, otherwise. Based on the experiments, a value of 2 or 3 time slices is ideal for the DNP dataset since there is a large drop in the XYT separation score when $\Delta T = 4$. Same is true for the OctoPisa dataset.

Effect of the *min_points* (κ) Parameter Recall that another parameter of the flock algorithm is *min_points*, which refers to the minimum objects that should consist a flock. Based on these experiments, we conclude that the selection of minimum number of points is the most trivial since a large value for min points

tends to produce no flock or few flocks; it is a tradeoff between having more flocks but with fewer members, or having few flocks but with more members.

Effect of the *radius* (ϵ) Parameter Lastly, we have also observed the effect of the *radius* parameter, which defines the spatial closeness among flock members. As observed in Figure 2, the number of flocks generally increases as ϵ increases. Meanwhile, the flock cohesion degrades (i.e., intra-distance increases) as ϵ increases. The XYT flock separation score tends to improve as ϵ increases when excluding the cases wherein the discovered flocks do not overlap in time (i.e., maximum XYT separation score is obtained) or no flocks were found. Meanwhile, the route similarity separation score generally improves as larger values of ϵ are used. As with previously observed parameters, the silhouette coefficients for varying ϵ remains close to 1.

The effect of *radius* as compared with the effect of the other parameters is as follows:

1. Out of all the scores used in assessing the effects of the parameters, the number of moving flocks has been the most sensitive. Generally, its value is directly proportional to the value of ϵ while it is inversely proportional to the other parameters. It is also worth noting that a higher number of moving flocks does not necessarily mean that the obtained flock results is better since the quality of the moving flocks may decrease when there are too many flocks discovered.
2. Compared to other flock validity measures, we consider flock cohesion as most important since it is in harmony with and explicit in the definition of a flock (i.e., a flock consists of members that are spatially close together over a specific time duration). While the flock cohesion score linearly increases (i.e., flock cohesion degrades) as the ϵ increases, the cohesion score did not change as much with respect to changing values of the other parameters. Thus, we can conclude that the *radius* has a larger impact on the obtained flock results compared to the other parameters. Excluding the cases wherein no flocks are discovered (i.e., the flock separation score is irrelevant) and the cases wherein there is no overlap in time among the discovered flocks (i.e., the XYT separation score is set to the maximum), higher ϵ generally improves the separation scores whereas higher values for the other parameters generally degrades the separation scores.
3. As a final point, the silhouette coefficient scores obtained by varying different parameters for both datasets were consistently close to 1, except for cases where less than 2 flocks were found.

Based on these experiments, we conclude that (1) The selection of minimum number of points is most trivial since a large value for *min_points* tends to a few flocks, if any at all; it is a tradeoff between having more flocks but with fewer members, or having few flocks but with more members. We also conclude that (2) the most crucial parameter is the *radius*, since it exhibited a larger effect on the flock cohesion score compared to the other parameters. Lastly, we

conclude that (3) while *radius* is the most crucial parameter, it is still important to choose good values for the other parameters since they still affect the quality of the discovered flocks.

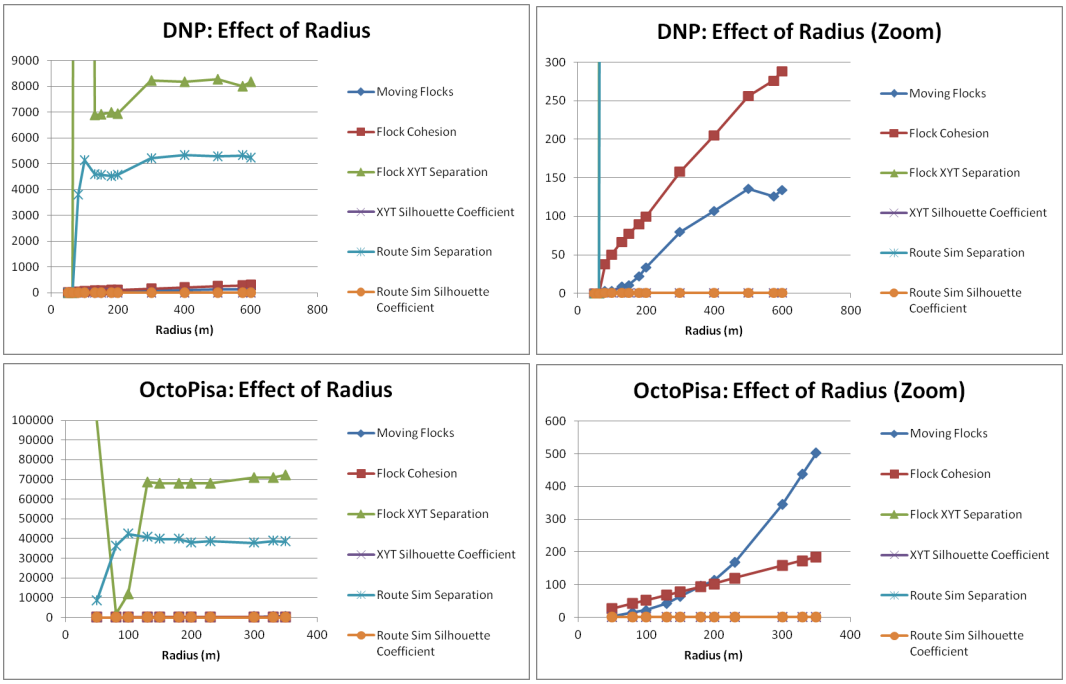


Fig. 2. Effect of the *radius* parameter with respect to flock quality measures in the two datasets.

4.3 Finding a Suitable *radius* Value

Since *radius* is a crucial parameter of the flock algorithm, we propose the following technique, which is an extension of the technique introduced for the *Eps* parameter of DBSCAN. Since DBSCAN deals with single n-dimensional data points while the flock algorithm deals with 3D data points (spatial component plus time) that are connected through object IDs, adjustments to their technique are necessary to accommodate the points linked by the same object IDs. The general idea of the extended technique is to compute the k-th distance among objects that co-occur in the same time instant where k is $min\_points - 1$ and k-th distance refers to the distance of a point from its k-th nearest neighbour. Once the k-th distances have been computed for each point, they are sorted in non-ascending order and plotted as a line graph. The portion in which there is a sudden decrease in the k-th distance suggests an upper bound for the *radius*

parameter of the algorithm. It is important to note that the trend of the plots in Figure 3 from right to left is as follows: increasing distance leads to the inclusion of more entities as members of discovered flocks. This becomes less apparent once we reach the portion of sudden decrease up to the leftmost part of the plots. Within this range, larger distance may lead to an increase in more entities included members in the discovered flocks, but this increase is very small. In fact, the leftmost part of the plots represent the cases wherein very large distance values no longer results in any increase of the entities. This happens when all entities are already member of one and the same flock since the chosen radius, which is based on the plots' k-th distance, is too large.

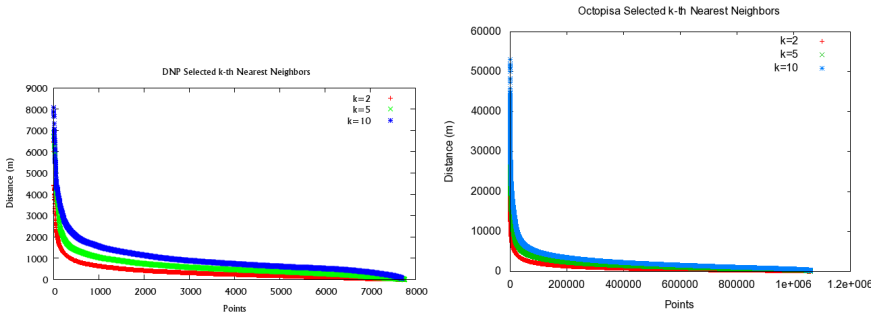


Fig. 3. Plot for k-th nearest neighbours for selected k's in the DNP (left) and the OctoPisa (right) datasets.

Using the top part of Figure 3 for the DNP dataset, a suggested *radius* value should be below the 500m-2000m range for flocks with at least 3 members (i.e., $k = 2$).

The obtained plot for the OctoPisa dataset is shown on the bottom part of Figure 3. It suggests 3000m-4000m as an upper bound for the *radius*. This is reasonable since the OctoPisa dataset covers a wider spatial area (about 4600 Km^2 vs about $48Km^2$ of DNP). It is also worth noting that the plots suggest different *radius* values for varying k's and yet, the division between the objects that would be included in some flock and those that are considered as noise is almost the same. Combining the suggested upper bound with contextual knowledge and the observation on the effect of *radius*, we recommend that a good range of values for *radius* is between 80m to 300m for DNP and between 50m to 300m for OctoPisa. Table 1 summarizes the main recommendations for a good range of parameter values for the two datasets.

5 Conclusions and Future Work

This paper provides an empirical evaluation of the effects of parameters in two different moving objects datasets, aimed at delineating a data-driven parameter estimation method for flock mining. We have evaluated the parameter setting

Parameter Name	*OctoPisa*	*DNP*	*Remarks*
κ	2-3	2-3	Prefer higher values but should consider number of discovered flocks, cohesion and separation scores
ΔT	3-4	2-3	Prefer higher values but consider number of discovered flocks, cohesion and separation scores
R	$<$ 4 mins. best: 1-2	$<$ 6 min. best: 1 & 4	Based on XYT separation score
ϵ	50m to 300m	80m to 300m	the DBSCAN-based plot (gives the optimal result in terms of cluster assignment) and the plots on moving flocks, cohesion and separation scores

Table 1. A table summarizing the main suggestions for flock parameters

methods in trajectories of pedestrian moving in a park and GPS data sets of moving vehicles.

We are also studying algorithm validation methods, which has been omitted here for lack of space. We are extending the experiments including additional trajectory datasets to further validate our results and to propose a formal framework for general flock mining parameters evaluation. Future work includes the extension of this approach to mining other kinds of moving objects such as animals.

References

1. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.
2. Fosca Giannotti and Dino Pedreschi, editors. *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*. Springer, 2008.
3. M. C. Gonzlez, C. A. Hidalgo, and A.-L. Barabsi. Understanding individual human mobility patterns. *Nature*, 453, 2008.
4. Joachim Gudmundsson and Marc J. van Kreveld. Computing longest duration flocks in trajectory data. In *GIS*, 2006.
5. Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases. *Proc. VLDB Endow.*, 1:1068–1080, 2008.
6. Patrick Laube, Stephan Imfeld, and Robert Weibel. Discovering relative motion patterns in groups of moving point objects. *International Journal of Geographical Information Science*, 19(6):639–668, 2005.
7. Monica Wachowicz, Rebecca Ong, Chiara Renso, and Mirco Nanni. Finding moving flock patterns among pedestrians through collective coherence. *International Journal of Geographical Information Science*, 25(11):1849–1864, 2011.

F2G: Efficient Discovery of Full-Patterns

Rui Henriques, Sara C. Madeira, and Cláudia Antunes

Dep. of Computer Science and Engineering
Instituto Superior Técnico, University of Lisbon, Portugal
{rmch,sara.madeira,claudia.antunes}@ist.utl.pt

Abstract. An increasing number of biomedical tasks, such as pattern-based biclustering, require the disclosure of the transactions (e.g. genes) that support each pattern (e.g. expression profiles). The discovery of patterns with their supporting transactions, referred as *full-pattern mining*, has been solved recurring to extensions over Apriori and vertical-based algorithms for frequent itemset mining. Although pattern-growth alternatives are known to be more efficient across multiple biological datasets, there are not yet adaptations for the efficient delivery of full-patterns. In this paper, we propose a pattern-growth algorithm able to discover full-patterns with heightened efficiency and minimum memory overhead. Results confirm that for dense datasets or low support thresholds, a common requirement in biomedical settings, this method can achieve significant performance improvements against its peers.

1 Introduction

The discovery of frequent patterns is increasingly accomplished in biological settings to identify orchestrations of genes and of their products using exhaustive and efficient searches [17]. However, biological tasks often require the output of both patterns and their supporting transactions. This task is referred as *full-pattern mining*. Illustrating, in gene expression analysis the interest is not so centered on the frequent expression profiles, but mainly on the group of genes that support those expression profiles. Full-pattern mining has been adopted in the past for biclustering [14, 17], gene association analysis [2] and integrative genomic studies [11] using gene expression data and genomic structural variations.

Existing full-pattern miners are simple extensions of frequent itemset mining algorithms that rely on bitset vectors to represent the supporting transactions per pattern [13, 17, 14]. However, bitset vectors offer efficiency problems in terms of memory and time for biological datasets with a medium-to-high number of records. Thus, the goal of this work is to study efficient alternatives for the full-pattern mining task. In particular, we propose the first variant of the FP-growth method, referred as F2G (Frequent Full-pattern Growth), able to deliver full-patterns with heightened efficiency. Experimental results hold evidence for its superior performance. We also show that existing scalability principles and alternative pattern representations can be easily included in F2G.

The paper is structured as follows. In *section 2*, the full-pattern mining task is formalized and its relevance is motivated. In *section 3*, the contributions and limitations from existing research are covered. The proposed F2G algorithm is described in *section 4*. Finally, key implications from the evaluation of F2G against state-of-the-art alternatives are synthesized in *section 5*.

2 Background

Let $\mathcal{L}$ be a finite set of items, and I be an itemset $I \subseteq \mathcal{L}$. A *transaction* t is a pair (t_{id}, I) with $t_{id} \in \mathbb{N}$. An *itemset database* D over $\mathcal{L}$ is a finite set of transactions. A transaction $t = (t_{id}, I)$ contains an itemset A, denoted $A \subseteq t$, if $A \subseteq I$. The *coverage* Φ_I of an itemset I is the set of all transactions in D in which the itemset I is contained: $\Phi_I = \{t \in D \mid I \subseteq t\}$. The *support* of an itemset I in D, denoted sup_I, is its coverage size $|\Phi_I|$.

A *full-pattern* is a pair (I, Φ_I), where I is an itemset and Φ_I the set of all transactions that contain I.

Given an itemset database D and a minimum support threshold θ, the ***full-pattern mining*** task consists of computing the set: $\{(I, \Phi_I) \mid I \subseteq \mathcal{L}, sup_I \geq \theta\}$.

For an illustrative itemset database D=$\{(t_1, \{a, c, e\}), (t_2, \{a, b, d\}), (t_3, \{a, c\})\}$, we have $\Phi_{\{a,c\}}$=$\{t_1, t_3\}$, $sup_{\{a,c\}}$=2. For a minimum support θ=2, the full-pattern mining task over D returns $\{(\{a\}, \{t_1, t_2, t_3\}), (\{a, c\}, \{t_1, t_3\})\}$.

2.1 Applications

Biclustering, the discovery of correlations among transactions (e.g. genes) based on a subset of overall items (e.g. conditions), is an increasingly popular biological task [12]. Fig.1 illustrates how biclustering relies on full-patterns.

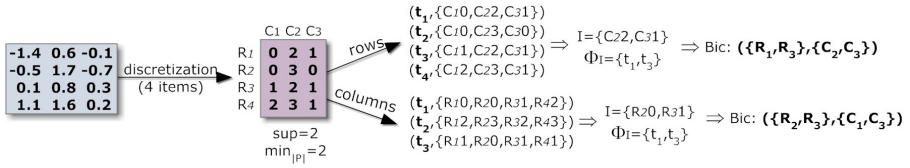


Fig.1: Mining constant biclusters over itemset databases using full-patterns.
The input dataset is discretized. To find biclusters with constant values on rows, the discrete value of each cell is concatenated with the respective column index. Each transaction corresponds to a row with these new values. Full-pattern mining is applied. The columns and rows for each bicluster are, respectively, derived from the items and supporting transactions of each full-pattern. Biclusters with constant values on columns can be mined using the transpose matrix.

Full-pattern mining methods to biclustering combine efficiency with flexibility [17]. This explains the increasing attention towards pattern-based approaches to biclustering such as BiModule [14], DeBi [17], RAP [16] and GenMiner [13].

Full-pattern mining has been also applied to perform association analysis using patient/gene/product-centric views and to study biological networks [2, 3]. These tasks have been applied over gene expression data, mutations and copy number variations [9], and discrete matrices derived from biological networks [3].

2.2 Related work

Separating the disclosure of transactions from the core mining task introduces a significant and unnecessary computational effort. Thus, the existing full-pattern miners rely on frequent itemset mining methods with implementations based on bitset vectors to represent the transaction-sets. Since these structures are maintained during all the search, is trivial to disclose the transactions shared per pattern at the end of the search. There are two approaches with this behavior.

A first approach combines the original Apriori-based strategy with the use of bitset vectors to track patterns coverage [1]. Illustrative implementations include LCM and CLOSE, used respectively by BiModule [14] and GenMiner [13] biclustering methods. They differ regarding the pruning methods applied over the itemset lattice of frequent candidates. Apriori-based methods generally suffer from the costs associated with the generation of a huge number of candidates for low support thresholds, a common requirement in biological tasks [14].

A second approach is the use of vertical-based methods, mostly through the extension of Eclat [19] and Carpenter [15]. Since vertical-based methods rely on intersection operations over transaction-sets to generate candidates, they require structures (such as bitset vectors or diffsets) to maintain the coverage per pattern. When the bitset cardinality becomes large, not only these structures consume a significant amount of memory, but also the intersection gets computationally costly. MAFIA [4] is an illustrative implementation adopted by DeBi [17]. However, these vertical-based methods are not competitive with horizontal-based methods for datasets with a number of transactions that significantly exceeds the average number of items [18].

2.3 The Problem

Existing full-pattern miners suffer from critical drawbacks. First, Apriori variants present efficiency problems for low support thresholds. Second, vertical-based approaches are not prone to deal with databases with a large set of transactions. Although FP-Growth method can overcome these problems [8], this method was not yet adapted to efficiently delivery full-patterns. A straightforward extension would be the annotation of each node on the underlying tree structure (FP-tree) with its supporting transactions, which undesirably leads to an impracticable additional computational complexity. Since most of the biological tasks rely on large and dense datasets and low support thresholds, the study of efficient FP-Growth extensions to discover full-patterns is of critical importance.

3 Solution

Since FP-Growth method relies on compact tree structures, it is positioned as an attractive alternative to tackle the computational overhead of the existing methods that maintain bitset vectors for every frequent candidate. Additionally, it neither requires candidate generation nor multiple database scans. In this section we propose a variant of the original FP-Growth method to deliver full-patterns with heightened efficiency. This algorithm is referred as *F2G*, *F*requent *F*ull-pattern *G*rowth. Similarly to the original FP-Growth method, F2G relies on a compact tree structure (FP-tree), which is recursively mined to enumerate all frequent patterns. Patterns are generated by concatenating the pattern suffixes with the frequent patterns discovered from conditional FP-trees where suffixes are removed. Suffixes are composed according to an ascending frequency order to prune the search space. In F2G, the transactions are optimally stored since they appear at most once in the FP-tree. Thus, unlikely the original method, the transaction-IDs are not lost at the very first scan.

Algorithm 1 describes F2G. The required adaptations, when taking FP-growth algorithm [8] as the basis of comparison, are highlighted by a gray background. To illustrate them consider the new (conditional) FP-trees represented in Fig.2, obtained from the application of Algorithm 1 over the itemset database D=$\{(t_0, \{A, C, D, F\}), (t_1, \{B, D, E, F\}), (t_2, \{B, D, F\}), (t_3, \{A, B, E\}), (t_4, \{A, F\}), (t_5, \{A, B, D, E, F\})\}$.

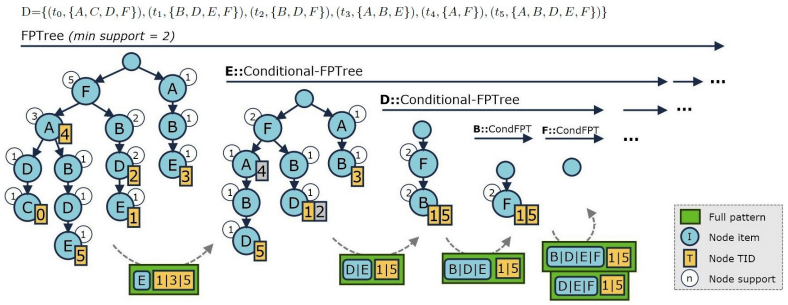


Fig.2: Illustrative behavior of F2G

The adaptations can be synthesized in three steps. *First*, the transaction-IDs are stored in the leaf node of an itemset path in the FP-tree without redundancy to guarantee optimal memory usage. To achieve this, the insertion of transactions in the FP-tree, addTransaction (*line 7*), is adapted so the identifier can be added to the item-node with least frequency. Considering the {F,A,D,C} path from the original FP-Tree in Fig.2, the identifiers t_0 and t_4 are placed in the item-nodes D and A. *Second*, before removing an infrequent node from a path (*line 16*), we need to exchange the transaction-IDs from the infrequent node to its parent (*line 17-19*). In this way the IDs assigned to the target suffixes rise from leafs to the root as long as shorter conditional FP-trees are built. Illustrating, since item C ($|\Phi_C|$=1) in the FP-tree from Fig.2 is infrequent (θ=2), t_0 is propagated upwards. *Third*, we recursively add the transaction-IDs assigned to the parental nodes of a frequent itemset to compute a full-pattern (*line 25*). Also, with the addition of a full-pattern, the identifiers need to be copied to the parental nodes (*line 30*). For the illustrative case, t_1 and t_5 are propagated towards its parent nodes during the construction of the E, D and B conditional FP-trees. The computational time added by these extensions is residual when compared with the construction of conditional trees (*lines 29, 35*).

Algorithm 2 describes the methods related with the (conditional) FP-tree construction. The computational impact of changing the methods addTransaction (*line 11*) and addPrefixPath (*line 19*) is residual. Note, additionally, that the order of items in the header list of the conditional FP-trees should preserve the order observed in the original FP-tree (*line 21*). This is due to the fact that a potential reordering of least-frequent items (different from the original ordering) may no longer preserve transactions at the end of itemset paths. To illustrate this constraint, consider the E conditional FP-tree from Fig.2. If we had opted to not maintain the order of the item nodes, B would appear below the root node, causing transaction t_5 to be relocated. This constraint has a residual impact on the performance. On one hand it has a slight deteriorate effect, since the items in

Algorithm 1: F2G Algorithm

```
   Output: FrequentFullPattern[] fullPatterns
1  Method: runFullPatternGrowthDiscovery
   Input: Transaction[] data, double support
2  Map<Int,Int> mapSup ← getItemsFrequency(data);
3  data ← removeInfrequentItems(data,mapSup);
4  data ← sortItemsets(data); //sort items in desc. freq. order
5  FPTree tree;
6  foreach Transaction trans : data do
7  |   tree.addTransaction(trans.itemset,trans.id) ;
8  tree.createHeaderList(mapSup) ;
9  F2G(tree, ∅, mapSup) ;

10 Method: F2G
   Input: FPTree tree, Itemset α, Map<Int,Int> mapSup
11 pruning(tree, α, mapSup); //FP-BONSAI optimization
12 if tree.hasSinglePath() then addAllCombForPath(tree.path, α) else
   FPGrowthMultiplePaths(tree, α, mapSup)

13 Method: FPGrowthMultiplePaths
   Input: FPTree tree, Itemset α, Map<Int,Int> mapSup
14 foreach Int item : tree.headerList /*items in reverse order*/ do
15 |   if mapSup[item] < relativeMinsup then
16 |   |   foreach Node node : tree.getItemNodes(item) do
17 |   |   |   node.parent.trans←node.parent.trans ∪ node.trans ;
18 |   |   |   node.trans = ∅ ;
19 |   |   continue;
20 |   β.values ← α ∪ item;
21 |   β.support ← min(α.support,mapSup[item]);
22 |   foreach Node node : tree.mapItemNodes.get(item) do
23 |   |   node.parent.trans ← node.parent.trans ∪ node.trans ;
24 |   |   β.trans ← β.trans ∪ node.trans ;
25 |   fullPatterns.add(β) ;
26 |   Path[] prefixPaths; //β cond. base (prefixes co-occuring with suffix pattern)
27 |   foreach Node node: tree.getItemNodes(item) do
28 |   |   Path path = node.getParentsUntilRoot();
29 |   |   path.trans ← node.trans ;
30 |   |   prefixPaths.add(path);
31 |   Map<Int,Int> mapβSup ← getItemsSup(prefixPaths);
32 |   FPTree βtree; //β conditional FP-Tree
33 |   foreach Path path : prefixPaths do
34 |   |   βtree.addPrefixPath(path, mapβSup, θ) ;
35 |   βtree.createHeaderList(mapβSup, tree.headerList) ;
36 |   if βtree.hasNodes() then F2G(βtree, β, mapβSup)

37 Method: addAllCombForPath //recursively adds path nodes with prefix
   Input: Path path, Itemset α
38 Node node ← path.retrieveFirst();
39 β.items ← α ∪ node.item;
40 β.support ← node.counter;
41 β.trans ← node.trans ;
42 fullPatterns.add(β) ;
43 if path.hasMoreNodes() then
44 |   addAllCombForPath(path, α);
45 |   addAllCombForPath(path, β);
```

Algorithm 2: FP-Tree Construction Methods

```
1  Method: addTransaction
   Input: Itemset itemset, int tid /*transaction ID*/
2  Node node ← root;
3  foreach Int item : itemset.getItems() do
4  |   if node.hasChild(item) then
5  |   |   Node newNode ← createNode(item, node /*parent*/);
6  |   |   node ← newNode;
7  |   else node ← node.getChild(item) if item==itemset.last() then node.trans ←
   |   node.trans ∪ tid

8  Method: addPrefixPath
   Input: Path path, Map<Int,Int> mapSup, Int θ /*support*/
9  Node transNode;
10 foreach Node node : path.nodes() /*backward order*/ do
11 |   if mapSup.get(node.item) < θ then continue ... /* code for adding a path to a
   |   FP-Tree */
12 |   transNode ← node;
13 transNode.trans ← transNode.trans ∪ path.getTransactions() ;

14 Method: createHeaderList
   Input: Map<Int,Int> mapSup, Int[] headerListSuper
15 headerList ← getItemNodes(). sortByIndexIn(headerListSuper) ;
```

the tree may not be locally order by frequency. On the other hand, this contraint is seized to optimize the efficiency of building conditional FP-trees.

Further Options: F2G method is compliant with further options that promote its scalability. In particular, we briefly motivate how F2G can comply with the adoption of compressed representations and parallelization principles.

A frequent itemset is *maximal* if is frequent and all supersets are infrequent, while is *closed* if is frequent and there exists no superset with the same support. FPMax and FPClose [6] use variants of conditional FP-trees to reduce the running time. These variants, Maximal/Closed Frequent Itemset trees, are optimally constructed to keep track of maximal/closed patterns. Since these trees preserve the item order of the original FP-tree header, the discovery of maximal/closed full-patterns can easily follow the extensions proposed in the F2G method. An alternative method is to rely on hybrid tree-projections [18].

Furthermore, parallelization and distribution principles proposed for FP-growth can be also applied for F2G to improve its scalability [7]. F2G comply with data partitioning principles [10] when relying on two steps. First, F2G is applied for each partition in order to retrieve sets of full-patterns. Second, the globally frequent itemsets are identified and their supporting transactions across partition merged. An alternative direction is to build a global FP-tree and to parallelize the (conditional) FP-tree construction methods [5], which also preserves the soundness of the F2G variant.

4 Results

In this section, we compare the performance of state-of-the-art implementations of Bitset Apriori and Eclat[1] with F2G on synthetic and real datasets. The algorithms are implemented under JVM version 1.6.0-24. The experiments were computed using an Intel Core i5 2.30GHz with 6GB of RAM.

[1] http://www.philippe-fournier-viger.com/spmf/

4.1 Synthetic datasets

The properties of the generated datasets are described in Table 1. Patterns with different shapes (varying number of transactions and items) were planted. The number of supporting transactions and items for each pattern follow an Uniform distribution over the ranges presented in Table 1. These settings were developed to mimic commonly observed biclusters in gene expression matrices.

Matrix size (♯rows × ♯columns)	500×50	1000×100	2000×200	4000×400
Nr. of hidden patterns	5	10	15	25
Nr. transactions for the hidden patterns	[10,14]	[14,30]	[30,50]	[50,100]
Nr. items for the hidden patterns	[5,7]	[6,8]	[7,9]	[8,10]
Minimum support assumption	θ=1%	θ=1%	θ=1%	θ=1%

Table 1: Properties of the generated dataset settings

The data density, the average percentage of total items per transaction, can significantly vary from 5% to 50% across biological settings. Therefore, we evaluate our approach in two steps. First, the density is fixed as 20%. The changes in performance with varying size are the focus. Second, a specific size is fixed and the density is varied from 5% to 33%.

In Fig.3 we assess the computational overhead of adapting the FP-Growth to perform full-pattern mining. F2G adds only a residual time and memory cost. The impact of sorting items in the header table is minor. Contrasting with F2G, the naive way of attaching and gathering transactions on the FP-tree nodes (extended FP-growth) strongly penalizes the performance.

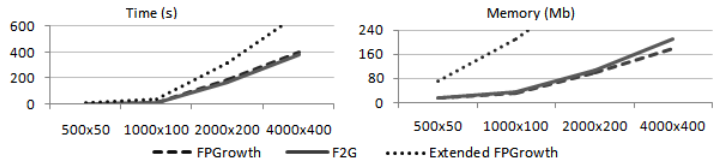


Fig.3: Efficiency of F2G and FPGrowth methods for datasets with varying size

Fig.4 compares the performance of efficient implementations of the major full-pattern miners for different data settings. F2G has significant gains in efficiency against Bitset Apriori and Eclat. The generation of candidate sets is penalized when the number of patterns is considerably high. The high-dimensionality of the datasets penalizes both the time and memory efficiency of Eclat.

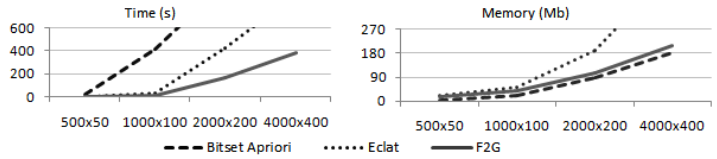


Fig.4: Efficiency of full-pattern mining methods for datasets with varying properties

To further compare the performance of full-pattern mining methods, we fixed the 1000×100 experimental setting and varied the level of sparsity. This analysis

is illustrated in Fig.5. First, F2G is the approach more able to deal with dense datasets. Second, the adoption of horizontal approaches for full-pattern mining is the best option in terms of memory for the generated settings.

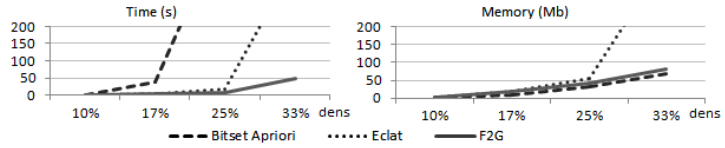


Fig.5: Efficiency of full-pattern mining methods for datasets with varying density

4.2 Real datasets

To assess the performance of the target approaches in real datasets, we adopted alternative gene expression datasets[2]. In particular, Fig.7 illustrates results for the yeast dataset for varying support thresholds. The same relative behavior can be observed across the remaining datasets from the selected repository. These datasets were discretized (assuming a Gaussian distribution of values and a target density of 10%) and, finally, column indexes were concatenated with items, as illustrated in Fig.1. Previous observations remain valid. F2G is the most efficient option in terms of time and, along with Apriori, a competitive choice for efficient memory usage.

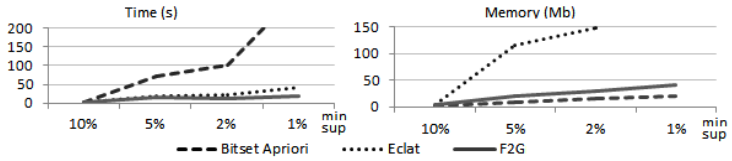


Fig.6: Efficiency of full-pattern mining methods for the yeast (2884×17) microarray

To evaluate the performance of full-pattern miners in different biological settings, we adopted alternative datasets following the procedures described in CP4IM project[3]. Fig.6 presents their performance on the primary-tumor dataset. Similarly, F2G offers the best compromise in terms of efficiency. The same relative behavior can be also observed across other UCI biological datasets, such as lymph and heart-cleveland[3].

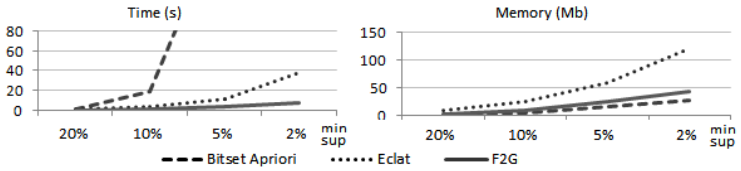


Fig.7: Efficiency of full-pattern mining methods for the primary-tumor dataset

[2] http://www.upo.es/eps/bigs/datasets.html
[3] http://dtai.cs.kuleuven.be/CP4IM/datasets/

5 Discussion

This work formalizes the full-pattern mining task and motivates its relevance for a critical set of biological applications. The performance of existing approaches is discussed. To tackle their inefficiencies, we propose the F2G algorithm, a pattern growth algorithm that discloses the supporting transactions per pattern with minimum space overhead and heightened efficiency.

F2G relies on FP-tree variants that store transaction-IDs without redundancy and on efficient propagation techniques. We show that F2G can easily accommodate principles from existing research to deliver alternative full-pattern representations or to discover full-patterns in distributed settings.

Results on both synthetic and real datasets show the superior performance of the proposed method. In particular, F2G delivers a distinctive performance both for dense and large datasets and for low support thresholds, common settings in biological tasks. To the best of our knowledge, this is the first attempt to compare full-pattern mining algorithms.

Acknowledgments

This work was supported by *Fundação para a Ciência e Tecnologia* under the research project D2PM, PTDC/EIA-EIA/110074/2009, and the PhD grant SFRH/BD/75924/2011.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB. pp. 487–499. Morgan Kaufmann, San Francisco, USA (1994)
2. Alves, R., R-Baena, D., Aguilar-Ruiz, J.: Gene association analysis: a survey of frequent pattern mining from gene expression data. B.Bioinf. 11(2), 210–224 (2010)
3. Bebek, G., Yang, J.: Pathfinder: mining signal transduction pathway segments from protein-protein interaction networks. BMC Bioinformatics 8 (2007)
4. Burdick, D., Calimlim, M., Gehrke, J.: Mafia: A maximal frequent itemset algorithm for transactional databases. In: ICDE. pp. 443–452. IEEE CS (2001)
5. Chen, D., Lai, C., Hu, W., Chen, W., Zhang, Y., Zheng, W.: Tree partition based parallel frequent pattern mining on shared memory systems. In: IPDPS. p. 8 (2006)
6. Grahne, G., Zhu, J.: Efficiently using prefix-trees in mining frequent itemsets. In: FIMI. CEUR W.Proc., vol. 90. CEUR-WS (2003)
7. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15(1), 55–86 (2007)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. 29(2), 1–12 (2000)
9. Hochreiter, S., Bodenhofer, U., Heusel, M., Mayr, A., Mitterecker, A., Kasim, A., Khamiakova, T., Van Sanden, S., Lin, D., Talloen, W., Bijnens, L., Göhlmann, H., Shkedy, Z., Clevert, D.A.: FABIA: factor analysis for bicluster acquisition. Bioinformatics 26(12), 1520–1527 (2010)
10. Javed, A., Khokhar, A.: Frequent pattern mining on message passing multiprocessor systems. Distributed and Parallel Databases 16(3), 321–334 (2004)
11. Lopez, F.J., Blanco, A., Garcia-Alcalde, F., Cano, C., Marin, A.: Fuzzy association rules for biological data analysis: A case study on yeast. BMC Bioinf. 9 (2008)

12. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Trans. Comput. Biol. Bioinformatics 1(1), 24–45 (2004)
13. Martinez, R., Pasquier, C., Pasquier, N.: Genminer: Mining informative association rules from genomic data. In: BIBM. pp. 15–22. IEEE CS (2007)
14. Okada, Y., Fujibuchi, W., Horton, P.: A biclustering method for gene expression module discovery using closed itemset enumeration algorithm. IPSJ Transactions on Bioinformatics 48(SIG5), 39–48 (2007)
15. Pan, F., Cong, G., Tung, A.K.H., Yang, J., Zaki, M.J.: Carpenter: finding closed patterns in long biological datasets. In: SIGKDD. pp. 637–642 (2003)
16. Pandey, G., Atluri, G., Steinbach, M., Myers, C.L., Kumar, V.: An association analysis approach to biclustering. In: SIGKDD. pp. 677–686. ACM (2009)
17. Serin, A., Vingron, M.: Debi: Discovering differentially expressed biclusters using a frequent itemset approach. Algorithms for Molecular Biology 6, 1–12 (2011)
18. Wang, J., Han, J., Pei, J.: Closet+: searching for the best strategies for mining frequent closed itemsets. In: SIGKDD. pp. 236–245. ACM (2003)
19. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: SIGKDD. pp. 326–335. ACM, New York, NY, USA (2003)

IndexSpan: Efficient Discovery of Item-Indexable Sequential Patterns

Rui Henriques, Sara C. Madeira, and Cláudia Antunes

Dep. of Computer Science and Engineering
Instituto Superior Técnico, University of Lisbon, Portugal
{rmch,sara.madeira,claudia.antunes}@ist.utl.pt

Abstract. The research on sequential pattern mining has been driven by efficiency principles. However, efficiency is still a critical drawback for tasks that require the discovery of sequential patterns with medium-to-large length. Many of these tasks, such as pattern-based biclustering, rely on datasets with item-indexable properties. An item-indexable database, typically observed in order-preserving datasets across biological and customer-service domains, does not allow item repetitions per sequence. In this work, we propose a new sequential pattern mining method, called IndexSpan, which is able to mine sequential patterns over item-indexable databases with heightened efficiency in comparison with the existing alternatives. The superior performance of IndexSpan is demonstrated on both synthetic and real datasets, and its relevance for multiple applications is discussed.

1 Introduction

During the last two decades, a few number of methods have been proposed to deal efficiently with the discovery of sequential patterns. These methods are prepared to deal with sequence databases with an arbitrary repetition of items per sequence. However, many real-world tasks rely on a more restricted form of sequences, item-indexable sequences, which assume simple ordering constraints among a subset of the overall items. To guarantee the consistency of ordering constraints among items, item-indexable sequences do not allow item repetitions. Illustrative examples of such databases include sequences derived from consumer ratings, shopping items ordered by the time of acquisition, schedule of tasks, order-preserving gene expression values in microarrays, among many others. Although some of these databases have varying levels of sparsity, item-indexable sequences derived from ratings and microarrays tend to be highly dense.

A common desirable property for the tasks formulated over these databases is the discovery of sequential patterns with a medium-to-large number of items. For instance, order-preserving patterns from ratings and biological data are only relevant above a minimum number of items. Although existing sequential pattern mining (SPM) approaches can be applied over these databases, they still show inefficiencies to deliver large patterns due to the combinatorial explosion of sequential patterns under low support thresholds [8]. Although there are principles for the discovery of colossal patterns based on the fusion of smaller itemsets [19], these approximations suffer from noise and, to our knowledge, have not

been extended for the SPM task. Additionally, the few dedicated methods able to discover sequential patterns in item-indexable databases [7, 8] show significant memory overhead.

This work proposes a new method for the efficient discovery of sequential patterns over item-indexable sequences. This is done by using efficient data structures that keep track of the position of items per sequence and by relying on fast database projections based on the relative order of items. Additional optimizations are proposed based on pruning techniques when the user has only interest in sequential patterns above a minimum length.

The paper is structured as follows. *Section 2* introduces and motivates the SPM task over item-indexable databases, and covers existing contributions with potential relevance for its solution. IndexSpan, the proposed SPM method prone to deal with item-indexable databases, is described in *section 3*. In *section 4*, the performance of IndexSpan is assessed on both real and synthetic datasets against peer algorithms. Finally, the implications of this work are synthesized.

2 Background

Let an item be an element from an ordered set $\mathcal{L}$. An *itemset* I is a set of non-repeated items, $I \subseteq \mathcal{L}$. A *sequence* s is an ordered set of itemsets. A sequence $a{=}<a_1...a_n>$ is a *subsequence* of $b{=}<b_1...b_m>$ ($a{\subseteq}b$), if $\exists_{1\leq i_1<..<i_n\leq m}$: $a_1 \subseteq b_{i_1},..,a_n \subseteq b_{i_n}$. A *sequence database* is a set of sequences $D = \{s_1, .., s_n\}$.

The illustrative sequence $s_1{=}<\{a\},\{be\}>{=}a(be)$ is contained in $s_2{=}(ad)c(bce)$ and is maximal w.r.t. to $D{=}\{ae, (ab)e\}$.

The **coverage** Φ_s of a sequence s w.r.t. to a set of sequences D, is the set of all sequences in D with s as subsequence: $\Phi_s = \{s' \in D \mid s \subseteq s'\}$. The **support** of a sequence s in D, denoted sup_s, is its coverage size $|\Phi_s|$.

To illustrate the previous concepts, consider the following sequence database $D = \{s_1 = (bc)a(abc)d, s_2 = cad(acd), s_3 = a(ac)c\}$. For this database, we have $|\mathcal{L}|{=}4$, $\Phi_{\{a(ac)\}} = \{s_1, s_2, s_3\}$, and $sup_{\{a(ac)\}}{=}3$.

Given a set of sequences D and some user-specified minimum support threshold θ, a sequence $s \in D$ is *frequent* when is subsequence of at least θ sequences. The **sequential pattern mining** (SPM) problem consists of computing the set of frequent sequences, $\{s \mid sup_s \geq \theta\}$.

The set of maximal frequent sequences for the illustrative sequence database, $D{=}\{(bc)a(abc)d, cad(acd), a(ac)c\}$, under a minimum support $\theta{=}3$ is $\{a(ac), cc\}$.

Let an item-indexable sequence be a sequence without repeated items. An item-indexable sequence database is a set of item-indexable sequences.

Let $|I|$ be the length of an itemset, and let the length of a sequence $|s|$ be the number of item occurrences, $\Sigma_i\, |s^i|$. Given a set of item-indexable sequences D, a minimum support threshold θ, and a minimum sequence length δ. The target task of **SPM over item-indexable sequences** consists of computing:

$$\{s \mid sup_s \geq \theta \wedge |s| \geq \delta\}$$

This formalization allows the definition of new methods prone to seize the properties of item-indexable sequences. Understandably, the resulting sequential patterns, referred as item-indexable sequential patterns, preserve the consistency of item ordering constraints since they do not allow for item duplicates.

2.1 Applications

Some of the most prominent applications for SPM task that rely on item-indexable databases include:

- order-preserving biclustering, a form of local clustering, that relies on sequential patterns [7, 3]. This type of biclustering is commonly applied over biological data, including gene expression and genomic structural variation analysis. To compose the database from real-value or discrete matrices, the column indexes are linearly ordered for each transaction according to their values. Each transaction is, consequently, seen as a sequence of items that correspond to column indexes. Biclusters are derived from the set of items and supporting transactions for each sequential pattern, as illustrated in Fig.1;
- pattern-centric analysis of recommendations based on user preferences and of service quality based on questionnaires [6]. In these scenarios, item-indexable sequences are derived from an ordering of user ratings. Interestingly, frequent precedences and co-occurrence disclose important priorities or similarities across the evaluated aspects for different sets of users;
- other key applications, such as frequent scheduling/planning, shopping, and traveling behavior [18, 5].

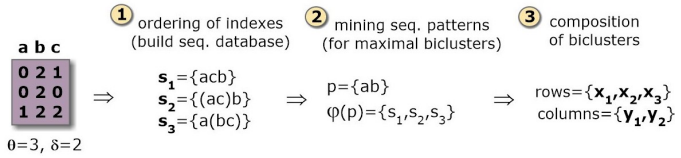


Fig.1: Mining order-preserving biclusters from item-indexable databases

2.2 Related Work

Although general SPM methods are not optimized to deal with item-indexable specificities, they have been the largely adopted to solve these applications [8]. Since the SPM problem proposal [1], multiple extensions and applications have been proposed, ranging from scalable implementations to alternative pattern representations. Current SPM methods can be classified into three main categories: apriori-based, pattern-growth, and early-pruning [9]. Apriori-based algorithms [13], and vertical-based variations [17], rely on join procedures to generate candidate sequences in a breadth-first manner using multiple database scans. To overcome the computational complexity of maintaining the support count for each sequence generated, alternatives such as the use of bitmaps or direct comparison have been proposed [4, 2].

Pattern growth methods [10, 11, 2] avoid the Apriori-based costs from candidate generation by building an expressive representation of the database and by recursively traversing it to grow the frequent sequences. PrefixSpan [10], one of the most efficient options to SPM, recursively constructs patterns by growing their prefix and by maintaining their corresponding postfix subsequences into

projected databases. In the absence of gap-based constraints, this guarantees a narrowed search space and avoids the generation of candidates since it only counts the frequency of local sequences. The major cost of PrefixSpan resides on the construction of projected databases.

Early-pruning methods [16, 4, 12] emerged more recently in the literature. They adopt a sort of position induction to prune candidate sequences very early in the mining process and to avoid support counting as much as possible. These algorithms usually employ a table to track the last positions of each item in the sequence to evaluate whether the item can be appended to a given prefix, thus avoiding support counting and generation of infrequent candidates.

The drawback of these SPM alternatives is that their performance does not scale for very low support thresholds, which is often required in item-indexable contexts to obtain medium-to-large sequential patterns. In fact, new methods can seize the item-indexable property, that guarantees that each item appears at most one time per item-indexable sequence, to minimize this problem.

Seizing this property, Liu and Wang [7, 8] proposed an alternative SPM method that constructs a compact tree structure, OPC-Tree, where sequences sharing the same prefix are gathered and recorded in the same branch. The discovery of frequent subsequences and the association of rows with frequent subsequences are performed simultaneously. However, the memory complexity of OPC-Tree is $\Theta(n \times m^2)$, where n is the number of records and m the average number of items per transaction. Although some pruning techniques can be applied in the OPC-Tree structure, their impact is not sufficient to turn this approach scalable for medium-to-large databases.

3 Solution: IndexSpan

To avoid the drawbacks of existing approaches, we propose the IndexSpan algorithm, an extension of PrefixSpan to discover sequential patterns with heightened efficiency from item-indexable sequence databases.

Comparison of existing SPM algorithms [9] shows key heuristics to turn the SPM efficient: mechanisms to reduce the support counting; narrowing of the search space; optimally sized data structure representations of the sequence database; and early pruning of candidate sequences. Seizing these properties, IndexSpan guarantees a search space as small as possible and relies on a narrow search procedure, depth-first search.

IndexSpan extends PrefixSpan [10] in order to incorporate additional efficiency gains from three principles. First, IndexSpan relies on an easily indexable and compacted version of the original sequence database. Second, it uses faster and memory-efficient database projections. A projected database only maintains a list with the IDs of the active sequences. Finally, IndexSpan relies on early-pruning techniques. IndexSpan is described in Algorithm 1.

IndexSpan considers the three following structural adaptations over the PrefixSpan algorithm. First, it maintains a simple matrix in memory that maintains the index of each item per row. This matrix is constructed at the very beginning (*lines 2-5*) and the original database is removed. Additionally, for sparse databases, this matrix can be replaced by a vector of hash tables to optimize memory usage. Considering an illustrative sparse database with $|\mathcal{L}|$=20,

Algorithm 1: IndexSpan

Input: sequence database D, minimum support θ, minimum sequence length δ
Output: set of sequential patterns S
Note: α is a sequence, D_α is the α-projected database
(D_α simply maintains a reference to the current sequences)

```
1  mainMethod() begin
2  |  foreach sequence s in D /*add array of item indexes per sequence*/ do
3  |  |  foreach item c do
4  |  |  |  s.indexes[c] ← position(s,c);
5  |  α.items ← φ; α.trans ← φ;
6  |  indexSpan(α,D);

7  indexSpan(α,Dα) begin
8  |  foreach frequent item c in Dα do
9  |  |  β.items ← α.items ∪ c; //co-occurrence (c is added to the last α itemset)
10 |  |  γ.items ← α.items · c; //α precedes c (c is inserted as a new itemset)
11 |  |  //pruning and fast gathering of supporting transactions (for efficient data
   |  |  projection)
12 |  |  foreach sequence s in Dα do
13 |  |  |  currentIndex ← s.indexes[c];
14 |  |  |  upperIndex ← s.indexes[αn] /*αn is the last item*/;
15 |  |  |  if leftPositions(currentIndex)≥δ-|α| /*pruning*/ then
16 |  |  |  |  if currentIndex > upperIndex then
17 |  |  |  |  |  γ.trans ← γ.trans ∪ s.ID;
18 |  |  |  |  else
19 |  |  |  |  |  if currentIndex=upperIndex ∧ c>αn then β.trans ← β.trans∪s.ID
20 |  |  if supβ(Dα) ≥ θ then
21 |  |  |  S ← S ∪ {β};
22 |  |  |  Dβ ← fastProjection(β,Dα);
23 |  |  |  indexSpan(β,Dβ);
24 |  |  if supγ(Dα) ≥ θ then
25 |  |  |  S ← S ∪ {γ};
26 |  |  |  Dγ ← fastProjection(γ,Dα);
27 |  |  |  indexSpan(γ,Dγ);

28 fastProjection(β,Dα) begin
29 |  foreach sequence s in Dα do
30 |  |  currentIndex ← s.indexes[βn];
31 |  |  upperIndex ← s.indexes[βn−1];
32 |  |  if leftPositions(currentIndex)≥δ-|α| /*pruning*/ then
33 |  |  |  if currentIndex > upperIndex then
34 |  |  |  |  Dβ ← Dβ ∪ s;
35 |  |  |  else
36 |  |  |  |  if currentIndex=upperIndex ∧ c > αn then Dβ ← Dβ ∪ s
37 |  return Dβ;
```

D=$\{a(cp), am, (ac)\}$. Instead of relying on a 20×3 matrix to track the indexes [[0 -1 1 ...][...][...]], it is enough to monitor the positions of the available items [h_1{a:0,c:1,p:1},h_2{a:0,m:1},h_3{a:0,c:0}] to obtain the index (e.g. h_2(m)=1).

These data structures support position induction. The idea behind is simple: if an item's last/start position precedes the current prefix/postfix position, the item can no longer appear before/after the current prefix.

Second, a projected database can be constructed with heightened efficiency by avoiding the need to update and maintain postfixes. A projected database simply maintains the identifiers of the supporting sequences for a specific prefix.

To know if a sequence is still frequent when an item is added over a specific prefix, there is only the need to compare its index against the index of the previous item as well as their lexical order for the case where the index is the same (i.e. the new item co-occurs with the last items of the pattern). In this way, database projections, the most expensive step of PrefixSpan both in terms of time

and memory, are handled with heightened efficiency. The proposed projection method is described in Algorithm 1, *lines 12-19* and *28-37*.

Finally, the input minimum number of items per sequential pattern, δ, can be used to prune the search as early as possible. If the number of items of the current prefix ($|\alpha|$) plus the items of a postfix s_α (computed based on the current and last index positions) is less than δ, then the sequence identifier related with the s_α postfix can be removed from the projected database since all the patterns supported by s will have a number of items below the inputted threshold.

For an optimal pruning, this assessment is performed before item indexes comparisons, which occurs in two distinct moments during the prefixSpan recursion (Algorithm 1 *lines 15* and *32*).

The efficiency gains from fast database projections and early pruning techniques, combine with the absence of memory overhead, turn IndexSpan highly attractive in comparison with the OPC-Tree peer method.

4 Results

In this section, we evaluate the performance of IndexSpan and competitive alternatives on synthetic and real datasets. IndexSpan was implemented in Java (JVM version 1.6.0-24). We adopted PrefixSpan[1], still considered a state-of-the-art SPM method, and the OPC-Tree method [7] as the bases of comparison. The experiments were computed using an Intel Core i5 2.30GHz with 6GB of RAM.

4.1 Synthetic datasets

The generated experimental settings are described in Table 1. First, we created dense datasets (each items occurs in every sequence) by generating matrices up to 2.000 rows and 100 columns. Each sequence is derived from the ordering of column indexes for a specific row according to the generated values, as illustrated in Fig.1 from Section 2. Understandably, each of the resulting item-indexable sequences contains all the items (or column indexes), which leads to a highly dense dataset. Sequential patterns were planted in these matrices by maintaining the order of values across a subset of columns for a subset of rows. The number and shape of the planted sequential patterns were also varied. The number of supporting sequences and items for each sequential pattern followed a Uniform distribution over the ranges presented in Table 1.

For each setting we instantiated 6 matrices, 3 matrices with a background of random values and 3 matrices with values generated according to a Gaussian distribution. The results are an average across these matrices. The properties of the generated databases and planted patterns are detailed in Table 1.

Fig.2 compares the performance of the alternative approaches for these settings in terms of time and maximum memory usage. Both PrefixSpan and OPC-Tree can be seen as competitive baselines to assess efficiency. Note that we evaluate the impact of mining sequential patterns in the absence and presence of the δ input, the minimum number of items per pattern, for a fair comparison.

[1] Implementation from SPMF: http://www.philippe-fournier-viger.com/spmf/

Matrix size (♯rows × ♯columns)	100×30	500×50	1000×75	2000×100
Nr. of hidden seq. patterns	5	10	20	30
Nr. rows for the hidden seq. patterns	[10,14]	[12,20]	[20,40]	[40,70]
Nr. columns for the hidden seq. patterns	[5,7]	[6,8]	[7,9]	[8,10]
Assumptions on the inputted thresholds	θ=5% δ=3	θ=5% δ=4	θ=5% δ=5	θ=5% δ=6

Table 1: Properties of the generated dataset settings

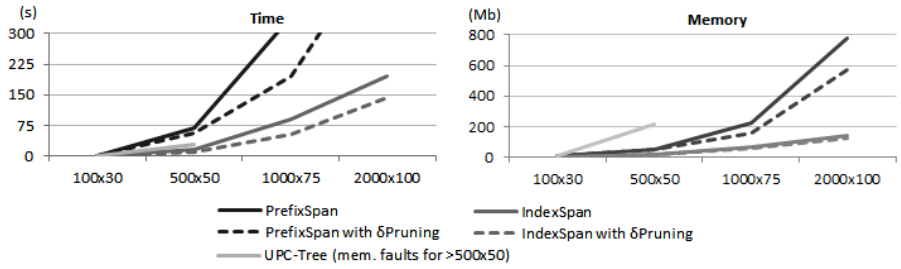


Fig.2: Performance of alternative SPM methods for datasets with varying properties.

Two main observations can be derived from this analysis. First, the gains in efficiency from adopting fast database projections are very significant. In particular, the adoption of fast projections for hard settings dictates the scalability of the SPM task. Pruning methods should also be considered in the presence of the pattern length threshold δ. Contrasting with OPC-Tree and PrefixSpan, IndexSpan guarantees acceptable levels of efficiency for matrices up to 2000 rows and 100 columns for a medium-to-large occupation of sequential patterns (∼3%-10% of matrix total area). Second, IndexSpan performs searches with minimal memory waste. The memory is only impacted by the lists of sequence identifiers maintained by prefixes during the depth-first search. Memory of PrefixSpan is slightly hampered due to the need to maintain the projected postfixes. OPC-Tree requires the full construction of the pattern-tree before the traversal, which turns this approach only applicable for small-to-medium databases. For an allocated memory space of 2GB, we were not able to construct OPC-Trees for input matrices with more than 40 columns.

To further assess the performance of IndexSpan, we fixed the 1000×75 experimental setting and varied the level of sparsity by removing specific positions on the input matrix, while preserving the planted sequential patterns. We randomly selected these positions to cause a heighten variance of length among the generated sequences. The amount of removals was varied between 0 and 40%. This analysis is illustrated in Fig.3.

Two main observations can be retrieved. First, to guarantee an optimal memory usage, there is the need to adopt vectors of hash tables in IndexSpan. Second, although the use of these new data structures hampers the efficiency of IndexSpan, the observable computational time is still significantly preferable over the PrefixSpan alternative.

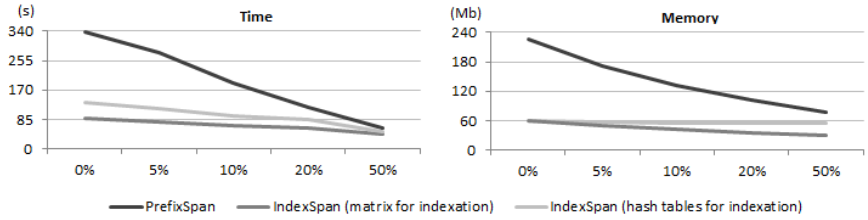


Fig.3: Performance for varying levels of sparsity (1000×75 dataset).

Finally, in order to assess the impact of varying the number of co-occurrences vs. precedences, we adopted multiple discretizations for the 1000×75 dataset. By decreasing the size of the discretization alphabet, we are increasing the amount of co-occurrences and, consequently, decreasing the number of itemsets per sequence. This analysis is illustrated in Fig.4. When the number of precedences per sequence is very small (<10), the efficiency tends to significantly decrease due to the exponential increase of sequential patterns. However, for the remaining discretizations, the efficiency does not strongly differ since the number of frequent patterns is identical and pattern-growth methods are able to deal with co-occurrences and precedences in similar ways (Algorithm 1 *lines 20-27*).

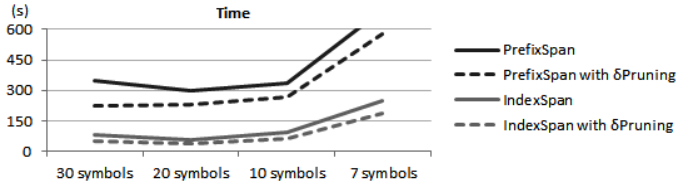


Fig.4: Performance for varying weights of precedences vs. co-occurrences.

4.2 Real datasets

To assess the performance of the target approaches in real datasets, we adopted multiple gene expression matrices[2]: dlblc (180 items per instance, 660 instances), coloncancer (62 items per instance, 2000 instances), leukemia (38 items per instance, 7129 instances). The goal is to discover order-preserved biclusters. For this purpose, we followed the procedure described in Fig.1 to generate the sequence databases using a discretization alphabet with 20 symbols. The positions corresponding to missing values were removed. Fig.5 compares the performance of the alternative approaches for the θ=8% and δ=5 thresholds. This analysis reinforces the previous observations. OPC-Tree is bounded by the size of the database. The adoption of IndexSpan strategies to deal with item-indexable se-

[2] http://www.upo.es/eps/bigs/datasets.html
http://www.bioinf.jku.at/software/fabia/gene_expression.html

quences strongly impacts the SPM performance, and, consequently, the ability to discover order-preserving biclusters in real data.

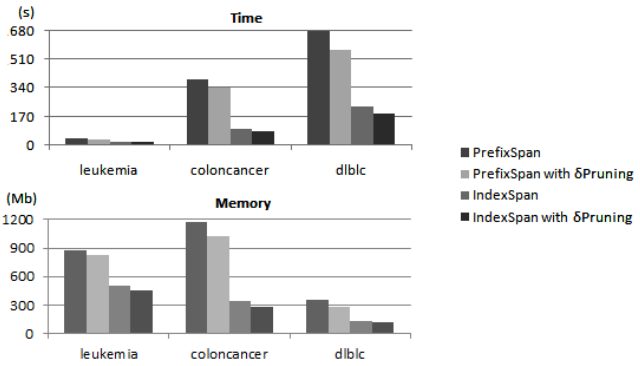


Fig.5: Performance of SPM-based order-preserving biclustering for biological data.

5 Discussion

This work formalizes the task of performing sequential pattern mining over item-indexable databases and motivates its relevance for a critical set of applications. The performance of existing approaches based on general SPM methods and on dedicated algorithms, such as the UPC-Tree, are discussed. To tackle the inefficiencies of existing solutions, we propose the IndexSpan algorithm.

IndexSpan relies on position induction to deliver fast and memory-free database projections. Additionally, early-pruning techniques with impact on the performance of IndexSpan can be adopted to guarantee that only large sequential patterns are discovered.

Since IndexSpan relies on a pattern-growth search, it can easily accommodate principles from existing research to deliver alternative pattern representations or to discover sequential patterns in distributed settings. Potential directions include the incorporation of principles of methods that extend PrefixSpan to discover condensed patterns, such as CloSpan [15], or to parallelize the mining task in distributed settings, such as MapReduce [14].

Results on both synthetic and real datasets show the superior performance of the proposed method. IndexSpan makes an optimal use of memory and is able to achieve significant improvements in computational time on both sparse and dense datasets.

Acknowledgments

This work was supported by *Fundação para a Ciência e Tecnologia* under the research project D2PM, PTDC/EIA-EIA/110074/2009, and the PhD grant SFRH/BD/75924/2011.

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE. pp. 3–14. IEEE CS, Washington, DC, USA (1995)
2. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: ACM SIGKDD. pp. 429–435. ACM, New York, NY, USA (2002)
3. Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z.: Discovering local structure in gene expression data: the order-preserving submatrix problem. In: RECOMB. pp. 49–57. ACM, New York, NY, USA (2002)
4. Chiu, D.Y., Wu, Y.H., Chen, A.L.P.: An efficient algorithm for mining frequent sequences by a new strategy without support counting. In: ICDE. pp. 375–. IEEE CS, Washington, DC, USA (2004)
5. Han, J., Yang, Q., Kim, E.: Plan mining by divide-and-conquer. In: ACM SIGMOD IW on Research Issues in DMKD (1999)
6. Kumar, P., Krishna, P., Raju, S.: Pattern Discovery Using Sequence Data Mining: Applications and Studies. Igi Global (2011)
7. Liu, J., Wang, W.: Op-cluster: Clustering by tendency in high dimensional space. In: ICDM. pp. 187–. IEEE CS, Washington, DC, USA (2003)
8. Liu, J., Yang, J., Wang, W.: Biclustering in gene expression data by tendency. In: Comput. Systems Bioinformatics Conf. pp. 182–193. IEEE (2004)
9. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surveys 43(1), 3:1–3:41 (2010)
10. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Trans. on Knowl. and Data Eng. 16(11), 1424–1440 (2004)
11. Pei, J., Han, J., Mortazavi-Asl, B., Zhu, H.: Mining access patterns efficiently from web logs. In: PADKK. pp. 396–407. Springer-Verlag, London, UK, UK (2000)
12. Salvemini, E., Fumarola, F., Malerba, D., Han, J.: Fast sequence mining based on sparse id-lists. In: ISMIS. pp. 316–325. Springer-Verlag, Berlin, Heidelberg (2011)
13. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT. pp. 3–17. Springer-Verlag, London, UK, UK (1996)
14. qing Wei, Y., Liu, D., shan Duan, L.: Distributed prefixspan algorithm based on mapreduce. In: Inf. Tech. in Medicine and Education. vol. 2, pp. 901–904 (2012)
15. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Datasets. In: SDM. pp. 166–177 (2003)
16. Yang, Z., Wang, Y., Kitsuregawa, M.: Lapin: effective sequential pattern mining algorithms by last position induction for dense databases. In: DASFAA. pp. 1020–1023. Springer-Verlag, Berlin, Heidelberg (2007)
17. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Mach. Learn. 42(1-2), 31–60 (2001)
18. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: IC on WWW. pp. 791–800. ACM, New York, NY, USA (2009)
19. Zhu, F., Yan, X., Han, J., Yu, P., Cheng, H.: Mining colossal frequent patterns by core pattern fusion. In: ICDE. pp. 706–715 (2007)

Sequential Pattern Mining from Trajectory Data

Elio Masciari[1], Gao Shi[2], and Carlo Zaniolo[2]

[1] ICAR-CNR – Institute of Italian National Research Council
masciari@icar.cnr.it
[2] UCLA – University of California Los Angeles
{zaniolo, gaoshi}@cs.ucla.edu

Abstract. In this paper, we study the problem of mining for frequent trajectories, which is crucial in many application scenarios, such as vehicle traffic management, hand-off in cellular networks, supply chain management. We approach this problem as that of mining for frequent sequential patterns. Our approach consists of a partitioning strategy for incoming streams of trajectories in order to reduce the trajectory size and represent trajectories as strings. We mine frequent trajectories using a sliding windows approach combined with a counting algorithm that allows us to promptly update the frequency of patterns. In order to make counting really efficient, we represent frequent trajectories by prime numbers, whereby the Chinese reminder theorem can then be used to expedite the computation.

1 Introduction

In this paper, we address the problem of extracting frequent patterns from trajectory data streams. Due to its many applications and technical challenges, the problem of extracting frequent patterns has received a great deal of attention since the time it was originally introduced for transactional data [1, 4]. For trajectory data the problem was studied in [3, 12]. The challenge posed by data stream systems and data stream mining is that, in many applications, data must be processed continuously, either because of real time requirements or simply because the stream is too massive for a store-now & process-later approach. However, mining of data streams brings many challenges not encountered in database mining, because of the real-time response requirement and the presence of bursty arrivals and concept shifts (i.e., changes in the statistical properties of data). In order to cope with such challenges, the continuous stream is often divided into windows, thus reducing the size of the data that need to be stored and mined. This allows detecting concept drifts/shifts by monitoring changes between subsequent windows. Even so, frequent pattern mining over such large windows remains a computationally challenging problem requiring algorithms that are faster and lighter than those used on stored data. Thus, algorithms that make multiple scans of the data should be avoided in favor of single-scan, incremental algorithms. In particular, the technique of partitioning large windows into slides (a.k.a. panes) to support incremental computations has proved very valuable in DSMS [11] and will be exploited in our approach. We will also make use of the following key observation: in real world applications there is an obvious difference between the problem of (i) finding new association rules, and (ii) verifying the continuous validity of existing rules. In order to tame the size curse of point-based trajectory representation, we propose to partition trajectories using a suitable regioning strategy. Indeed, since trajectory data carry information with a detail not often necessary in many application scenarios, we can split the search space in regions having the suitable granularity and represent them as simple strings. The sequence of regions (strings) define the trajectory traveled by a given object. Regioning is a common assumption in trajectory data mining [9, 3] and in our case it is even more suitable since our goal is to extract typical routes for moving objects as needed to answer queries such as: *which are the most used routes between Los Angeles and San Diego?* thus extracting a pattern showing every point in a single route is useless.

The partitioning step allow us to represent a trajectory as string where each substring encodes a region, thus, our proposal for incremental mining of frequent trajectories is based on an efficient algorithm for frequent string mining. As a matter of fact, the extracted patterns can be profitably used in systems devoted to traffic management, human mobility analysis and so on. Although a real-time introduction of new association rules is neither sensible nor feasible, the on-line verification of old rules is highly desirable for two reasons. The first is that we need to determine immediately when old rules no longer holds to stop them from pestering users with improper recommendations. The second is that every window can be divided in small panes on which the search for new frequent patters execute fast. Every pattern so discovered can then be verified quickly. Therefore, in this paper we propose a fast algorithm, called *verifier* henceforth, for verifying the frequency of previously frequent trajectories over newly arriving windows. To this end, we use sliding windows, whereby a large window is partitioned into smaller panes[11] and a response is returned promptly at the end of each slide (rather than at the end of each large window). This also leads to a more efficient computation since the frequency of the trajectories in the whole window can be computed incrementally by counting trajectories in the new incoming (and old expiring) panes.

Our approach in a nutshell. As trajectories flow we partition the incoming stream in windows, each window being partitioned in slides. In order to reduce the size of the input trajectories we pre-process each incoming trajectory in order to obtain a smaller representation of it as a sequence of regions. We point out that this operation is well suited in our framework since we are not interested in point-level movements but in trajectories shapes instead. The regioning strategy we exploit uses PCA to better identify directions along which we should perform a more accurate partition disregarding regions not on the principal directions. The rationale for this assumption is that we search for frequent trajectories so it is unlikely that regions far away from principal directions will contribute to frequent patterns (in the following we will use frequent patterns and frequent trajectories as synonym). The sequence of regions so far obtained can be represented as a string for which we can exploit a suitable version of well known frequent string mining algorithms that works efficiently both in terms of space and time consumption. We initially mine the first window and store the frequent trajectories mined using a tree structure. As windows flow (and thus slides for each window) we continuously update frequency of existing patterns while searching for new ones, This step require an efficient method for counting (a.k.a verification). Since trajectories data are ordered we need to take into account this feature. We implement a novel verifier that exploits prime numbers properties in order to encode trajectories as numbers and keeping order information, this will allow a very fast verification since searching for the presence of a trajectory will result in simple (inexpensive) mathematical operations.

Remark. In this paper we exploit techniques that were initially introduced in some earlier works [14, 13, 15]. We point out that in this work we improved those approaches in order to make them suitable for sequential pattern mining. Moreover, data mining approaches validity relies in their experimental assessment, in this respect the experiments we performed confirmed the validity of the proposed approach.

2 Trajectory size reduction

For transactional data a tuple is a collection of features, instead, a trajectory is an ordered set (i.e., a sequence) of timestamped points. We assume a standard format for input trajectories, as defined next. Let P and T denote the set of all possible (spatial) positions and all timestamps, respectively. A trajectory Tr of length n is defined as a finite sequence $s_1, \cdots, s_n$, where $n \geq 1$ and each s_i is a pair (p_i, t_i) where $p_i \in P$ and $t_i \in T$. We assume that P and T are discrete domains, however this assumption does not affect the validity of our approach. For continuous locations, a viable approach is to

partition the space into regions in order to map the initial locations into discrete regions labeled with a timestamped symbol. The problem of finding a suitable partitioning for both the search space and the actual trajectory is a core problem when dealing with spatial data. Every technique proposed so far, somehow deals with regioning and several approaches have been proposed such as partitioning of the search space in several regions of interest (RoI)[3] and trajectory partitioning (e.g. [10]) by using polylines. In this section, we describe the application of Principal Component Analysis (PCA)[6] in order to obtain a better partitioning. Indeed, PCA finds *preferred* directions for data being analyzed. We refer as *preferred* directions the (possibly imaginary) axes where the majority of the trajectories lie. Once we detect the preferred directions we perform a partition of the search space along these directions. Due to space limitations, instead of giving a detailed description of the mathematical steps implemented in our prototype we will present an illustrating (real life) example, that will show the main features of the approach.

Example 1. Consider the set of trajectories depicted in Fig. 1(a) regarding bus movements in the Athens metropolitan area. There are several trajectories close to the origin of the axes (that is located at city downtown) so it is difficult to identify the most interesting areas for analysis.

(a) Original Data (b) Regions defined by exploiting IPCA

Fig. 1. Trajectory Pre-Elaboration steps

In order to properly assign regions we need to set a suitable level of granularity by defining the initial size s of each region, i.e. their diameter. In order to keep an intuitive semantic for regions of interest we partition the search space into square regions along the directions set by the eigenvalues returned by IPCA. Since the region granularity will affect further analysis being performed the choice of region size s is guided by *DBScan* an unsupervised density based clustering algorithm. The output of the regioning step is depicted in Figure 1(b).

Definition 1 (Dense Regions). *Let T be a set of trajectories, and X_I and Y_I the axes defined by IPCA, $C = \{C_1, C_2, \cdots, C_n\}$ a set of regions obtained with density based algorithm (DBScan) laying on X_I and Y_I, the regions defined by C_i's boundaries are* Dense.

3 Frequent Trajectories Mining

The regioning schema presented in previous section allows a compact representation of trajectories by the sequences of regions crossed by each trajectory, i.e. as a set of strings, where each substring encodes a region. It is straightforward to see that this representation transform the problem of searching frequent information in a (huge) set of multidimensional points into the problem of searching frequent (sub)strings in a set of strings representing trajectories. We point out that our goal is to mine frequent trajectories tackling the "where is" problem, i.e. we are interested in movements made by

objects disregarding time information (such as velocity). Moreover, since the number of trajectories that could be monitored in real-life scenarios is really huge we need to work on successive portion of the incoming stream of data called *windows*. Let $T = \{T_1, \cdots, T_n\}$ be the set of regioned trajectories to be mined belonging to the current window; T contains several trajectories where each trajectory is a sequence of regions. Let $S = \{S_1, \cdots, S_n\}$ denotes the set of all possible (sub)trajectories of T. The *frequency* of a (sub)trajectory S_i is the number of trajectories in T that contain S_i, and is denoted as $Count(S_i, T)$. The *support* of S_i, $sup(S_i, T)$, is defined as its frequency divided by the total number of trajectories in T. Therefore, $0 \leq sup(S_i, T) \leq 1$ for each S_i. The goal of frequent trajectories mining is to find all such S_i, whose support is greater than (or equal to) some given minimum support threshold α. The set of frequent trajectories in T is denoted as $\mathcal{F}_\alpha(T)$. We consider in this paper frequent trajectories mining over a data stream, thus T is defined as a sliding window over the continuous stream. Each window either contains the same number of trajectories (count based or physical window), or contains all trajectories arrived in the same period of time (time-based or logical window). T moves forward by a certain amount by adding the new slide (δ^+) and dropping the expired one (δ^-). Therefore, the successive instances of T are shown as $W_1, W_2, \cdots$. The number of trajectories that are added to (and removed from) each window is called its *slide size*. In this paper, for the purpose of simplicity, we assume that all slides have the same size, and also each window consists of the same number of slides. Thus, $n = |W|/|\mathcal{S}|$ is the number of slides (a.k.a. panes) in each window, where $|W|$ denotes the window size and $|\mathcal{S}|$ denotes the size of the slides.

Mining trajectories in W. As we obtain the string representation of trajectories, we focus on the string mining problem. In particular, given a set of input strings, we want to extract the (unknown) strings that obey certain frequency constraints. The frequent string mining problem can be formalized as follows. *Given a set T of input strings a given frequency threshold α , find the set S_F s.t. $\forall s \in S_F, count(s, T) > \alpha$*

Many proposal have been made to tackle this problem [7, 2]. We exploit in this paper the approach presented in [7]. The algorithm works by searching for frequent strings in different databases of strings, in our paper we do not have different databases, we have different windows instead. We first briefly recall the basic notions needed for the algorithm, more details can be found in [7, 2].

The suffix array SA of a string s is an array of integers in the range [1.. n], which describes the lexicographic order of the n suffixes of s. The suffix array can be computed in linear time [7]. In addition to the suffix array, we define the inverse suffix array SA^{-1}, which is defined by $SA^{-1}[SA[i]] = i \; \forall 1 \leq i \leq n$. The LCP table is an array of integers which is defined relative to the suffix array of a string s. It stores the length of the longest common prefix of two adjacent suffixes in the lexicographically ordered list of suffixes. The LCP table can be calculated in $O(n)$ from the suffix array and the inverse suffix array. The ω-interval is the longest common prefix of the suffixes is s. The algorithm is reported in Figure 2 and its features can be summarized as follows.

Function $extractStrings$ arrange the the input strings in the window W_i in a string S^{aux} consisting of the concatenation of the strings in W_i, using # as a separation symbol and \$ as termination symbol. Functions *buildSuffixes* and *buildPrefixes* computes respectively the suffixes and prefixes of S^{aux} and store them using SA and LCP variables. Function *computeRelevantStrings* first compute the number of times that a string s occurs in W_i and then subtract so called correction terms which take care of multiple occurrences within the same string of W_i as defined in [7]. The output frequent strings are arranged in a tree structure that will be exploited for incremental mining purposes as will be explained in next section.

Incremental Mining of Frequent Trajectories. As the trajectories stream flows we need to incremental update the frequent trajectories pattern so far computed (that are inserted in a *Trajectory Tree* (TT)). Our algorithm always maintains a union of the frequent trajectories of all slides in the current window W in TT, which is guaranteed

Method: MineFrequentStrings
Input: A window slide S of the input trajectories;
Output: A set of frequent strings S_F.
Vars:
A string S_{aux};
A suffix array SA;
A prefix array LCP.
1: $S^{aux} = extractStrings(S)$;
2: $SA = buildSuffixes(S^{aux})$;
3: $LCP = buildPrefixes(S^{aux})$;
4: $S_F = computeRelevantStings(W_0, SA, LCP)$
5: **return** S_F;

Fig. 2. The frequent string mining algorithm

to be a superset of the frequent pattern over W. Upon arrival of a new slide and expiration of an old one, we update the true count of each pattern in TT, by considering its frequency in both the expired slide and the new slide. To assure that TT contains all patterns that are frequent in at least one of the slides of the current window, we must also mine the new slide and add its frequent patterns to TT. The difficulty is that when a new pattern is added to TT for the first time, its true frequency in the whole window is not known, since this pattern wasn't frequent in the previous $n-1$ slides. To address this problem, we uses an auxiliary array (aux) for each new pattern in the new slide. The aux array stores the frequency of a pattern in each window starting at a particular slide in the current window. In other words, the auxiliary array stores frequency of a pattern for each window, for which the frequency is not known. The key point is that this counting can either be done eagerly (i.e., immediately) or lazily. Under the laziest approach, we wait until a slide expires and then compute the frequency of such new patterns over this slide and update the aux arrays accordingly. This saves many additional passes through the window. The pseudo code for the the algorithm is given in Figure 3. At the end of each slide, it outputs all patterns in TT whose frequency at that time is $\geq \alpha \dot{n} \dot{|}S|$. However we may miss a few patterns due to lack of knowledge at the time of output, but we will report them as delayed when other slides expire. The algorithm starts when the first slide has been mined and its frequent trajectories are stored in TT.

Herein, function $updateFrequencies$ updates the frequencies of each pattern in TT if it is present in S. As the new frequent patterns are mined (and stored in TT'), we need to annotate the current slide for each pattern as follows: if a given pattern t already existed in TT we annotate S as the last slide in which t is frequent, otherwise (t is a new pattern) we annotate S as the first slide in which t is frequent and create auxiliary array for t and start monitoring it. When a slide expires (denote it S_{exp}) we need to update the frequencies and the auxiliary arrays of patterns belonging to TT if they were present in S_{exp}. Finally, we delete auxiliary array if pattern t has existed since arrival of S and delete t, if t is no longer frequent in any of the current slides.

Algorithm Properties

Correctness This follows immediately from the fact that a pattern t belongs to $\mathcal{F}_\alpha(W)$ (the frequent patterns is a window), only if it also belongs to $\cap_i \mathcal{F}_\alpha(S_i)$ (the union of frequent patterns for each slide i in the window). Thus, every frequent pattern in W must show up after mining of at least one of the slides and then we add it to TT.

Max Delay The maximum delay allowed by our algorithm is $(n-1)$ slides. Indeed, after expiration of $(n-1)$ slides, we will have a complete history of the frequency of all frequent patterns of $W$and can report them. Moreover, the case in which a pattern is reported after $(n-1)$ slides of time, is rare. For this to happen, patterns support in

Method: IncrementalMaintenance
Input: A trajectory stream T.
Output: A trajectory pattern tree T_T.
Vars:
A window slide S of the input trajectories;
An auxiliary array aux;
A trajectory tree TT'

```
1:  For Each New Slide S_new
2:     updateFrequencies(TT, S);
3:     TT' = MineFrequentStrings(S_new);
4:     For Each trajectory t ∈ TT ∩ TT'
5:        annotateLast(S_new, t);
6:     For Each trajectory t ∈ TT' \ TT
7:        update(TT, t);
8:        annotateFirst(S_new, t, t.aux);
9:  For Each Expiring Slide S_exp
10:    For Each trajectory t ∈ TT
11:       conditionalUpdateFrequencies(S_exp, t);
12:       conditionalUpdate(t.aux);
13:       if t has existed since arrival of S
14:          delete(t.aux);
15:       if t no longer frequent in any of the current slides
16:          delete(t);
```

Fig. 3. The incremental miner algorithm

all previous $n-1$ slides must be less than α but very close to it, say $\alpha \cdot |S| - 1$, and suddenly its occurrence goes up in the next slide to say β, causing the total frequency over the whole window to be greater than the support threshold. Formally, this requires that $(n-1) \cdot (\alpha \cdot |S| - 1) + \beta \geq \alpha n |S|$, which implies $\beta = n + \alpha \cdot |S| - 1$. This is not impossible, but in real-world such events are very rare, especially when n is a large number (i.e., a large window spanning many slides).

Time Complexity The main steps of the algorithm are the counting phase of all patterns of TT in the new slide and the expired one (i.e. delta maintenance), and the computation (and insertion in TT) of new frequent patterns. Let us denote the average time for the counting step as $f(|S|, |TT|)$. f is a function of the size of TT ($|TT|$) and slide size ($|S|$). The time for computing new patterns (denote it $M(|S|, \alpha)$ is a function of the slide size and the threshold value α. The total running time to process each window will be $2 \cdot f(|S|, |TT|) + M(|S|, \alpha)$ up to some negligible terms (the factor 2 take into account the verification step for new and expiring slides). It is worth noticing that, the only part of this running time that depends on window size ($|W|$) is $|TT|$. Finally, the number of frequent patterns is significantly smaller than the $n \cdot \mathcal{F}_\alpha(S_i)$ since most frequent patterns are common between slides.

Space Consumption The memory required for running our algorithm, consists of an binary tree containing the new slide, and the pattern tree containing the frequent patterns so far computed (which is significantly smaller than the slides size). The only concern which remains is that we need an auxiliary array (of size $n-1$) for each pattern which has been added to TT recently (i.e., within the last n slides). After that period we no longer need an auxiliary array for that pattern and we release its memory. Therefore, the worst case happens when all patterns need such an array resulting in $4 \cdot n \cdot |TT|$ bytes of extra memory (assuming we use 4-byte integers for storing the frequency); this is not prohibitive since the number of patterns is not supposed to be very large.

A very fast verifier for trajectories. In the following, we first define the verifier notion and propose our novel verifier for trajectories data.

Definition 2. *Let T be a trajectories database, P be a given set of arbitrary patterns and min_{freq} a given minimum frequency. A function f is called a verifier if it takes T, P and min_{freq} as input and for each pattern $p \in P$ returns one of the following results: a) p's true frequency in T if it has occurred at least min_{freq} times or otherwise; b) reports that it has occurred less than min_{freq} times (frequency not required in this case).*

It is important to notice the subtle difference between verification and simple counting. In the special case of $min_{freq} = 0$ a verifier simply counts the frequency of all $p \in P$, but in general if $min_{freq} > 0$, the verifier can skip any pattern whose frequency will be less than min freq. This early pruning can be done by the Apriori property or by visiting more than $|T| - min_{freq}$ trajectories. Also, note that verification is different (and weaker) from mining. In mining the goal is to find all those patterns whose frequency is at least min_{freq}, but verification simply verifies counts for a given set of patterns, i.e. verification does not discover additional patterns. Therefore, we can consider verification as a concept more general than counting, and different from (weaker than) mining. The challenge is to find a verification algorithm, which is faster than both mining and counting algorithms, since the algorithm for extracting frequent trajectories will benefit from this efficiency. In our case the verifier needs to take into account the sequential nature of trajectories so we need to count really fast while keeping the right order for the regions being verified. To this end we exploit an encoding scheme for regioned trajectories based on some peculiar features of prime numbers.

4 Encoding Paths for Efficient Counting and Querying

A great problem with trajectory sequential pattern mining is to control the exponential explosion of candidate trajectory paths to be modeled because keeping information about ordering is crucial. Indeed, our regioning step heavily reduce the dataset size so the number of regions we have to deal with is of hundreds of regions instead of thousands of points. Since our approach is stream oriented we also need to be fast while counting trajectories and (sub)paths. To this end, prime numbers exhibit really nice features that for our goal can be summarized in the following two theorems. They have also been exploited for similar purposes for RFID tag encodings [8], but in that work the authors did not provide a solution for paths containing cycles as we do in our framework.

Theorem 1 (The Unique Factorization Theorem). *Any natural number greater than 1 is uniquely expressed by the product of prime numbers.*

As an example consider the trajectory $T_1 = ABC$ crossing three regions A,B,C. We can assign to regions A, B and C respectively the prime numbers 3,5,7 and the position of A will be the first ($pos(A) = 1$), the position of B will be the second ($pos(B) = 2$), and the position of C will be the third ($pos(C) = 3$). Thus the resulting value for T_1 (in the following we refer to it as P_1) is the product of the three prime numbers, $P_1 = 3 * 5 * 7 = 105$ that has the property that does not exist the product of any other three prime numbers that gives as results 105.

As it is easy to see this solution allows to easily manage trajectories since containment and frequency count can be done efficiently by simple mathematical operations. Anyway, this solution does not allow to distinguish among ABC, ACB, BAC, BCA, CAB, CBA, since the *trajectory number* (i.e. the product result) for these trajectories is always 105. To this end we can exploit another fundamental theorem of arithmetics.

Theorem 2 (Chinese Remainder Theorem). *Suppose that $n_1, n_2, \cdots, n_k$ are pairwise relatively prime numbers. Then, there exists W (we refer to it as* witness*) between 0 and $N = n_1 \cdot n_2 \cdots n_k$ solving the system of simultaneous congruences: $W\%n_1 = a_1$, $W\%n_2 = a_2$, ..., $W\%n_k = a_k$.*

Then, by Theorem 2, there exists W_1 between 0 and $P_1 = 3 * 5 * 7 = 105$. In our example, the witness W_1 is 52 since $52\%3 = 1 = pos(A)$ and $52\%5 = 2 = pos(B)$ and $52\%7 = 3 = pos(C)$. We can compute W_1 efficiently using the extended Euclidean algorithm. From the above properties it follows that in order to fully encode a trajectory (i.e. keeping the region sequence) it suffices to store two numbers its prime number product (we refer to it as trajectory number) and its witness. As a nice side-effect in order to obtain any information about region positions in the trajectory we can test with the maximum efficiency containment relationships (a simple division) and order checking (a sequence of divisions). In order to assure that no problem will arise in the encoding phase and witness computation we assume that the first prime number we choose for encoding is greater than the trajectory size. So for example if the trajectory length is 3 we encode it using prime numbers 5,7,11. A devil's advocate may argue that multiple occurrences of the same region leading to cycles violate the injectivity of the encoding function. To this end the following example will clarify our strategy.

Dealing with cycles. Consider the following trajectory $T_2 = ABCAD$, we have a problem while encoding region A since it appears twice in the first and fourth position. We need to assure that the encoding value of A is such that we can say that both $pos(A) = 1$ and $pos(A) = 4$ hold(we do not want two separate encoding value since the region is the same and we are interested in the order difference). Assume that A is encoded as $(41)_5$ (i.e. 41 on base 5, we use 5 base since the trajectory length is 5)) this means that A occurs in positions 1 and 4. The decimal number associated to it is $A = 21$, and we chose as the encoding for $A = 23$ that is the first prime number greater than 21. Now we encode the trajectory using $A = 23$, $B = 7$, $C = 11$, $D = 13$ thus obtaining $P_2 = 23023$ and $W_2 = 2137$ (since the remainder we need for A is 21). As it easy to see we are still able to properly encode even trajectories containing cycles. As a final notice we point out that the above calculation is made really fast by exploiting a parallel algorithm for multiplication operation. We do not report here the pseudo code for the encoding step explained above due to space limitations. Finally, one may argue that the size of prime numbers could be large, however in our case it is bounded since the number of regions is small as confirmed by several empirical studies [5](always less than a hundred of regions for real life applications we investigated).

Definition 3 (Region Encoding). *Given a set $R = \{R_1, R_2, \cdots, R_n\}$ of regions, a function enc from R to $\mathcal{P}$ (the positive prime numbers domain) is a* region encoding function *for R.*

Definition 4 (Trajectory Encoding). *Let $T_i = R_1, R_2 \cdots R_n$ be a regioned trajectory. A trajectory encoding ($E(T_i)$) is a function that associates T_i with a pair of integer numbers $\langle P_i, W_i \rangle$ where $P_i = \prod_{1..n} enc(R_i)$ is the trajectory number and W_i is the witness for P_i.*

Once we encode each trajectory as a pair $E(T)$ we can store trajectories in a binary search tree making the search, update and verification operations quite efficient since at each node we store the $E(T)$ pair. It could happen that there exists more than one trajectory encoded with the same value P but different witnesses. In this case, we store once the P value and the list of witnesses saving space for pointers and for the duplicate P's value. Consider the following set of trajectories along with their encoding values (we used region encoding values: $A = 5$, $B = 7$, $C = 11$, $D = 13$, $E = 15$): (ABC, $\langle 385, 366 \rangle$), (ACB, $\langle 385, 101 \rangle$), (BCDE, $\langle 15015, 3214 \rangle$), (DEC, $\langle 2145, 872 \rangle$). ABC and ACB will have the same P value (385) but their witnesses are $W_1 = 366$ and $W_2 = 101$, so we are still able to distinguish them.

Claim. The proposed trajectory encoding scheme performs a one-to-one encoding for any trajectory.

The proof easily follows by Theorem 2 by injectivity of prime numbers conversion properties.

Method: $checkContainment$
Input:
Two trajectories encodings $E(T_1)$ and $E(T_2)$;
Output:
Yes if $T_1 \in T_2$, no otherwise;
Method:
1: **if** $P_1 > P_2$ **return NO**
2: **if** $P_2 \% P_1 = 0$ then
3: **for each** R_i^1
4: **if** $pos((next(R_i^1))^2) < pos(R_i^2)$ **return NO**
5: **return YES**

Method: $updateTree$
Input:
a trajectory tree B_T and a new window NEW;
Output:
the updated version of B_T;
Vars:
a tree node N;
Method:
1: **for each** $T_i \in NEW$
2: $N = depthSearch(E(T_i))$
3: **if** $N \neq null$
4: $B_T = updateFrequency(N, E(T_i))$
5: **else** $insertNode(E(T_i))$
6: **if** $memoryneeded\, deleteOlderNode(B_T)$
7: **return** B_T

Fig. 4. Algorithms for checking containment and update tree

Once computed the encoded trajectories we build our synopsis by storing them in a search binary tree associated to the input trajectories. At each node of the binary tree we store the trajectory number, the witness and the frequency count for each trajectory. The insertion, remove and update operations have the usual meaning.

A simple algorithm for checking ordered containment. In order to perform efficient trajectory verification we should be able to check containment relationships between trajectories. More in detail, given two trajectories, we call them T_1 and T_2, and their encodings $E(T_1)$ and $E(T_2)$ we need to answer this question 'Is T_1 a sub-sequence of T_2?'. To this end we can exploit the mathematical features of encoded trajectories to design a simple and effective algorithm. In the following we denote the index of a region $R_i \in T_i$ as $pos(R_i)$ and the region following R_i in T_i as $next(R_i)$, we recall that we do not need to store the region positions since they can be obtained by simple math calculation on witnesses. If a region R_i appears into two different trajectories T_l and T_m we denote it as R_i^l and R_i^m.

The algorithm first checks if P value of T_1 is a divider of P value of T_2, if the latter holds this means that all the regions belonging to T_1 also belongs to T_2. To test if containment holds we need to check that every pair of consecutive regions in T_1 are also consecutive in T_2. Consider again our toy example and suppose to check wether $T_1 = BC$ is contained in $T_2 = ABC$. we have that $P_2 = 385$ and $P_1 = 77$. We test that $P_2 \% P_1 = 0$, now we have that $pos(B^2) = 2 < 3 = pos(C^2)$ thus the answer to the containment test is YES. In order to build a fast verifier for trajectory frequencies, we store an additional information at each node, i.e. the frequency of the trajectories stored at that node. Obviously if we have more than one witness for a given P value we will store the frequencies for each witness. Again, doing this we prevent new nodes insertion thus saving memory space. The initial tree is built for the first window W_1, we call it B_T. As new trajectories arise in the stream we continuously update it. In particular, for every incoming trajectory $T_i \in NEW$ we search the node N it belongs to by performing a $depthSearch$ on B_T of the encoding values for T ($E(T_i)$). If such a node N exists (this means that T_i is frequent) we update its frequency. In particular we update the frequency of the witness W_i for T_i (we recall that there may exists more than one witness for the same P value). If the trajectory does not exist in the tree we insert the corresponding node (annotating its timestamp). Finally, as the stream flows if we need to release memory we delete the older nodes (i.e. the ones with older timestamps).

# input sequences	Our Algorithm			T-Patterns		
	times	# regions	# patterns	times	# regions	# patterns
10,000	1.412	94	62	4.175	102	54
20,000	2.115	98	71	6.778	107	61
50,000	3.876	96	77	14.206	108	67
100,000	7.221	104	82	30.004	111	73

(a)

# input sequences	Our Algorithm			T-Patterns		
	times	# regions	# patterns	times	# regions	# patterns
10,000	1.205	94	41	4.175	102	37
20,000	2.003	98	50	6.778	107	43
50,000	3.442	96	59	14.206	108	49
100,000	6.159	104	65	30.004	111	58

(b)

Table 1. Performances comparison against static RoI

The proposed verifier is faster and require less memory than a traditional FP-tree that keeps a separate path for each trajectory. Moreover it improves the conditionalization step since it is implicitly performed when encoding trajectories values.

5 Experimental Results

In this section we will show the experimental results for our algorithms. We used the GPS dataset[16](this dataset being part of *GeoLife*project). It records a broad range of users outdoor movements, thus, the dataset allows a severe test for our frequent sequential pattern mining. In order to compare the effectiveness of our approach we compared it with *T-Patterns* system described in [3]. In particular since *T-Patterns* does not offer streaming functionalities we compare our system using a single large window and compare the extracted patterns w.r.t. the regioning performed. More in detail we compare our results w.r.t. the Static and Dynamic regioning offered by *T-Patterns*.

Comparison against Static RoI. In the following, we compare our algorithm against *T-Patterns* with static RoI by measuring the execution times, the number of extracted regions and the number of extracted patterns for a given support value. Table 1(a) and (b) summarize respectively the results obtained on sets of 10,000 up to 100,000 trajectories extracted for the GPS dataset with 0.5% and 1% as min support value. Table 1(a) shows that when the number of input trajectories increases the execution times linearly increases and our execution time is lower than *T-Patterns*. This can be easily understood since we exploit a really fast frequent miner. A more interesting result is the one on number of extracted regions. As explained in previous sections, we exploit PCA and we focus on regions along principal directions, this allow us to obtain less region but more informative since many trajectories will cross them. As a consequence having a smaller number of accurate regions allows more patterns to be extracted as confirmed in Table 1(a). The intuition behind this result is that when considering a smaller number of regions this imply a greater number of trajectories crossing those regions. The above features are confirmed by the results reported in Table 1(b) for 1% minimum support value (obviously it will change the execution times and number of patterns while the number of extracted regions is equal to previous table). Interestingly enough, the execution times for our algorithm slightly decrease as the min support value increases and this is due to the advantage we get by the verification strategy.

Comparison against Dynamic RoI. In the following, we compare our algorithm against *T-Patterns* with dynamic RoI by measuring the execution times, the number of extracted regions and the number of extracted patterns for a given support value. Tables 2(a) and (b) summarize respectively the results obtained on sets of of 10,000 up to 100,000 trajectories extracted for the GPS dataset with 0.5% and 1% as min support value. Also for this comparison, Table 2(a) shows, for 0.5% minimum support value, that when the number of input trajectories increases the execution times linearly increases and our execution time is better than *T-Patterns*. The other improvements obtained with our algorithm have the same rationale explained above. These features are confirmed by the results reported in 2(b) for 1% minimum support value.

Verifier Effectiveness. As mentioned above *T-Pattern* does not support streaming functionalities so we will report here only our performances on trajectory streams. We first report the results for our verifier using different number of input patterns. Table 3 summarizes the results we obtained on GPS dataset. For this experiment, we provided

# input sequences	Our Algorithm			T-Patterns		
	times	# regions	# patterns	times	# regions	# patterns
10,000	1.412	94	62	4.881	106	56
20,000	2.115	98	71	7.104	111	66
50,000	3.876	96	77	15.306	112	69
100,000	7.221	104	82	302.441	115	75

(a)

# input sequences	Our Algorithm			T-Patterns		
	times	# regions	# patterns	times	# regions	# patterns
1,000	1.205	94	41	5.002	105	40
2,000	2.003	98	50	7.423	108	46
5,000	3.442	96	59	15.974	113	53
10,000	6.159	104	65	32.558	116	60

(b)

Table 2. Performances comparison against dynamic RoI

Running Times	# Patterns
910	10,000
1,172	15,000
1,654	20,000
2,883	50,000
3,756	100,000

(a)

Running Times	# Patterns
634	10,000
972	15,000
1,231	20,000
2,033	50,000
2,977	100,000

(b)

Table 3. Verifier Effectiveness Results

a predefined set of patterns to verify, we first set minimum support to 0 thus verification coincides with counting. We vary the number of patterns given to the algorithms as shown in Table 3(a). In addition we run the same experiment while setting various minimum support value, the results are reported in Table 3(b) only for 0.5% value due to space limitations.

It is worth noticing that the proposed verifier is extremely fast and that the execution time is less than linear w.r.t. the number of input patterns. Such a result is relevant since in many practical situations, including those where data arrival rate is very high, continuously mining the data set is either impractical or unfeasible. For such cases, using the fast verifier allows to monitor the data stream in order to (i) confirm the validity of existing patterns, and (ii) detect any occurrence of concept-shift. In fact, our experiments suggest that concept-shift is always associated with a significant number ($> 5 - 10\%$) of frequent patterns becoming infrequent. Therefore, only when such changes are observed we need to call on a mining algorithm, which will discover the new patterns.

Mining Algorithm Performances. In this section we report the results we ran to test the performances of the proposed incremental mining algorithm for large sliding windows. At the best of our knowledge our algorithm is the first proposal for dealing with frequent pattern mining on trajectory streams so we do not have a "gold" standard to compare with, however the results obtained are really satisfactory since the running times are almost insensitive to the window size. We recall that the algorithm goal is maintaining frequent trajectories over large sliding windows. Indeed, the results shown in Table 4(a) show that the delta-maintenance based approach presented here is scalable with respect to the window size. Finally, we report the total number of frequent pattern as windows flow (the results shown in Table 4(b)). They are computed using a window size of 10,000 trajectories) for a minimum support value of 0,1 %. Indeed we report the total number of patterns that have been frequent wether they are still frequent or not, this information is provided to take into account the concept shift for data being monitored. The results in Table 4(b) shows that after 200 windows being monitored the number of patterns that resulted frequent in some window is more than doubled this means that the users habits heavily changed during the two years period.

6 Conclusion

In this paper we tackled this problem by introducing a very fast algorithm to verify the frequency of a given set of sequential patterns. The fast verifier has been exploited in order to solve the sequential pattern mining problem under the realistic assumption that we are mostly interested in the new/expiring patterns. This delta-maintenance approach effectively mines very large windows with slides, which was not possible before. In summary we have proposed an approach highly efficient, flexible, and scalable to solve the frequent pattern mining problem on data streams with very large windows. Our

Running Times	*Windows size*
773	10,000
891	25,000
1,032	50,000
1,211	100,000
1,304	500,000
2,165	1,000,000

(a)

# Window	*# Patterns*
1	85
10	106
20	125
50	156
100	189
200	204

(b)

Table 4. Mining Algorithm Results

work is subject to further improvements in particular we will investigate: 1) further improvements to the regioning strategy; 2) refining the incremental maintenance to deal with maximum tolerance for delays between slides.

7 Acknowledgments

The authors would like to thank Barzan Mozafari for the invaluable discussions that originated this work.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.
2. Johannes Fischer, Volker Heun, and Stefan Kramer. Optimal string mining under frequency constraints. In *PKDD*, pages 139–150, 2006.
3. Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *KDD*, pages 330–339, 2007.
4. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, 2000.
5. H. Jeung, M. Lung Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.
6. I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics, 2002.
7. A. Kügel and E. Ohlebusch. A space efficient solution to the frequent string mining problem for many databases. *Data Min. Knowl. Discov.*, 17(1):24–38, 2008.
8. C-H Lee and C-W Chung. Efficient storage scheme and query processing for supply chain management using rfid. In *SIGMOD08*, pages 291–302, 2008.
9. J-G Lee, J. Han, X. Li, and H. Gonzalez. *TraClass*: trajectory classification using hierarchical region-based and trajectory-based clustering. *PVLDB*, 1(1):1081–1094, 2008.
10. J-G Lee, J. Han, and K-Y Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD07*, pages 593–604, 2007.
11. J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 34(1):39–44, 2005.
12. Y. Liu, L. Chen ., J. Pei, Q. Chen, and Y. Zhao. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. In *PerCom*, pages 37–46, 2007.
13. E. Masciari. Trajectory clustering via effective partitioning. In *FQAS*, pages 358–370, 2009.
14. Elio Masciari. Warehousing and querying trajectory data streams with error estimation. In *DOLAP*, pages 113–120, 2012.
15. B. Mozafari, H. Thakkar, and C. Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In *ICDE*, pages 179–188, 2008.
16. Y. Zheng, Q. Li, Y. Chen, and X. Xie. Understanding mobility based on gps data. In *UbiComp 2008*, pages 312–321, 2008.

Extending ReliefF for Hierarchical Multi-label Classification ⋆

Ivica Slavkov[1], Jana Karcheska[2], Dragi Kocev[1], Slobodan Kalajdziski[2], and Sašo Džeroski[1]

[1] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2] Ss. Cyril and Methodius University, Faculty of Computer Science and Engineering, Skopje, Macedonia
{ivica.slavkov,dragi.kocev,saso.dzeroski}@ijs.si,
j.karceska@gmail.com,
slobodan.kalajdziski@finki.ukim.mk

Abstract. In the recent years, the data available for analysis in machine learning is becoming very high-dimensional and also structured in a more complex way. This emphasises the need for developing machine learning algorithms that are able to tackle both the high-dimensionality and the complex structure of the data. Our work in this paper, focuses on extending a feature ranking algorithm that can be used as a filter method for specific type of structured data. More specifically, we adapt the RReliefF algorithm for regression, for the task of hierarchical multi-label classification (HMC). We evaluate this algorithm experimentally in a filter-like setting by employing PCTs for HMCs as a classifier and we consider datasets from various domains. The results show that HMC-ReliefF can identify the relevant features present in the data and produces a ranking where they are among the top ranked.

Keywords: feature selection, feature ranking, feature relevance, structured data, hierarchical multi-label classification, multi-label classification, ReliefF

1 Introduction

The current trend in machine learning is that the data available for analysis is becoming increasingly more complex. The complexity arises both from the data being high-dimensional and from the data being more structured. On one hand, high-dimensional data presents specific challenges for many machine learning algorithms, especially with the stability of the produced results [8]. On the other, mining complex data and extracting knowledge from it has been identified as one of the most challenging problems in machine learning [4], [13].

For dealing with the high-dimensionality of the data various feature selection methods exist. They usually precede the induction of predictive models and can be classified as filter, wrapper and embedded methods [7]. Filter methods [2] are the simplest ones and they usually involve a feature ranking algorithm that produces a list of relevant

⋆ The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint first authors.

features. Wrapper methods [11] rely on classification algorithms to perform feature selection and are computationally expensive. Embedded methods [7] are basically classification algorithms that have the feature selection embedded in the model induction phase.

Learning in a supervised context, where the target is structured, has also attracted much attention. Several algorithms that were previously employed only for classification or regression purposes, have been extended to also work with structured targets. These include decision trees for hierarchical targets [19], SVMs for multi-label and hierarchical multi-label problems [6], as well as tree ensembles that can be additionally employed for vectors of multiple targets [10].

Our work in this paper focuses on tackling the feature selection problem in the context of structured targets. We consider this a relevant problem in machine learning that relates to both of the previously discussed data trends. So far, structured prediction has not been considered in the context of a feature ranking method and we consider this an novel and interesting line of work to pursue.

More specifically, we focus on the ReliefF [16] algorithm for feature ranking. This algorithm is instance based and it works in a very intuitive fashion, making it relatively easy to extend. Its theoretical properties have been extensively explored [16] and it is very successful in detecting relevant feautres in a dataset.

We extend ReliefF for a specific type of structured problems, namely those from the Hierarchical Multi-Label Classification (HMC) domain [17]. The target that is predicted for these problems is defined with a hierarchy of classes and each instance in the dataset can be labelled with more than one class at a time. By definition, when an instance is labelled with one class it is also labelled with all of its parent classes according to the given hierarchy.

This type of problems appear in different domains, for example in biology for the task of gene function prediction. Namely, for this task, each gene can be annotated by multiple functions and the functions are organised into a tree-shaped hierarchy or a directed acyclic graph such as the Gene Ontology [1]. Predicting the function of a certain gene, would have to take into account the multi-label annotation of each gene and also the hierarchical connections of these labels.

Considering this, we present the details of our work in the rest of this paper, organised as follows. In Section 2 we define more formally the HMC setting and present the distance measures appropriate for this setting. Then, in Section 3, we discuss in depth the original RReliefF algorithm for regression and explain our HMC-ReliefF extension of the algorithm. We present our experimental evaluation of the proposed HMC-ReliefF algorithm in Section 4. At the end, in Section 5, we present our conclusion and discuss further directions of work.

2 Hierarchical Multi-label Classification

As previously discussed, in our work we extend the ReliefF algorithm for the task of hierarchical multi-label classification (HMC). Hierarchical classification differs from traditional classification that the classes are organised in a hierarchy. An example that belongs to a given class automatically belongs to all its super-classes (this is known

as the *hierarchy constraint*). Furthermore, if an example can belong simultaneously to multiple classes that can follow multiple paths from the root class, then the task is called hierarchical multi-label classification (HMC) [19], [17].

We formally define the hierarchical multi-label classification setting as follows:

- A description space X that consists of tuples of values of primitive data types (discrete or continuous), i.e., $\forall X_i \in X, X_i = (x_{i_1}, x_{i_2}, ..., x_{i_D})$, where D is the size of the tuple (or number of descriptive variables),
- a target space S, defined with a class hierarchy $(C, \leq_h)$, where C is a set of classes and $\leq_h$ is a partial order (e.g., structured as a rooted tree) representing the superclass relationship ($\forall\, c_1, c_2 \in C : c_1 \leq_h c_2$ if and only if c_1 is a superclass of c_2),
- a set E, where each example is a pair of a tuple and a set, from the descriptive and target space respectively, and each set satisfies the hierarchy constraint, i.e., $E = \{(X_i, S_i) | X_i \in X, S_i \subseteq C, c \in S_i \Rightarrow \forall c' \leq_h c : c' \in S_i, 1 \leq i \leq N\}$ and N is the number of examples in E ($N = |E|$)

Calculating the distance between two different instances of the target space S_1 and S_2, can be done in different ways. In [19] the hierarchy of labels is represented as a vector of binary values. The vector is created by traversing the tree or DAG that is representing the hierarchy in pre-order and assigning a 0 or 1 sequentially in the vector for a missing or present label respectively.

This representation allows two hierarchies of labels for two instances to be compared by simply comparing two binary vectors. In our HMC-ReliefF algorithm we use a weighted Euclidean distance measure given with the following equation:

$$d(v_1, v_2) = \sqrt{\sum_i w(c_i)(v_{1,i} - v_{2,i})^2}, \tag{1}$$

where v_1 and v_2 are the binary vector representation of S_1 and S_2 respectively.

For example, consider the toy class hierarchy shown in Figure 1(a,b), and two data examples: (X_1, S_1) and (X_2, S_2) that belong to the classes $S_1 = \{c_1, c_2, c_{2.2}\}$ (boldface in Figure 1(b)) and $S_2 = \{c_2\}$, respectively. We use a vector representation with consecutive components representing membership of class c_1, c_2, $c_{2.1}$, $c_{2.2}$ and c_3, in that order (preorder traversal of the tree of class labels). The distance is then calculated as follows:

$$d(S_1, S_2) = d([1,1,0,1,0], [0,1,0,0,0]) = \sqrt{w_0 + w_0^2}.$$

The weighting function $w(c)$ allows for the hierarchical structure of the classes to be taken into account by making the value dependent on the depth of the hierarchy:

$$w(c) = w_0^{depth(c)}, 0 < w_0 < 1. \tag{2}$$

This scheme ensures that the differences higher in the hierarchy have bigger influence on the total distance.

If the hierarchy is represented with a DAG, this scheme needs to be modified. In this case more than one path from the root to a given class exists and with a different depth. This problem is solved with the following recursive equation:

$$w(c) = w_0 \cdot avg(w(parent_j(c))). \tag{3}$$

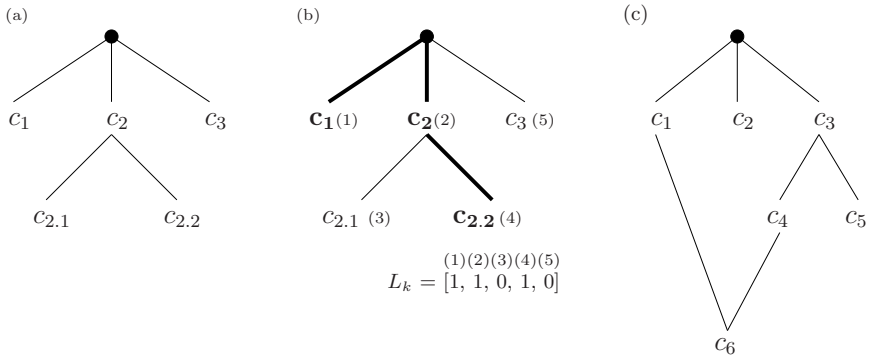


Fig. 1: Toy examples of hierarchies structured as a tree and a DAG.(a) Class label names contain information about the position in the hierarchy, e.g., $c_{2.1}$ is a subclass of c_2. (b) The set of classes $S_1 = \{c_1, c_2, c_{2.2}\}$, shown in bold in the hierarchy, represented as a vector (L_k). (c) A class hierarchy structured as a DAG. The class c_6 has two parents: c_1 and c_4.

By using this weighting function, the weight of the different possible parents is averaged and in [19] this is recommended as a good way to take into account multiple inheritance which occurs in DAGs.

3 HMC-ReliefF Algorithm

The Relief family of algorithms are instance based methods for estimating the feature relevance. The original Relief algorithm was proposed in [9] and is limited to two-class classification problems. The algorithm was extended in [12] to deal with multi-class problems. The extension was named ReliefF. Later, it was also adapted for regression problems [15] and named RReliefF.

In general, the feature relevance value awarded by the Relief algorithm is an approximation of the following difference of probabilities [12]:

$$\begin{aligned} W[F] = P(\text{diff. value of } F | \text{nearest inst. from diff. class}) - \\ P(\text{diff. value of } F | \text{nearest inst. from same class}) \end{aligned} \tag{4}$$

In the case of classification, the basic intuition behind the ReliefF algorithm is to estimate the relevance of a feature according to how well it distinguishes between neighbouring instances. If the feature has different values for neighbouring instances that are of different class (nearest miss), then it is awarded a higher relevance values. However, if the values of the class for the neighbouring instances are the same (nearest hit), then the relevance value is decreased.

Although the hierarchical multi-label setting is a classification one, extending the ReliefF algorithm is not a good idea. Namely, if we simply treat two instances annotated by different parts of the hierarchy in a simple hit/miss scenario, we would simply

Algorithm 1 Pseudocode for the RReliefF algorithm, taken from [16].

Input: for each training instance a vector of feature values $\mathbf{x}$ and predicted value $\tau(\mathbf{x})$
Output: the vector W of estimations of the relevance of features

1: set all N_{dC},$N_{dF}[F]$,$N_{dC\&dF}[F]$,$W[F]$ to 0
2: **for** $i = 1$ to m **do**
3: randomly select and instance R_i
4: select k instances I_j nearest to R_i
5: **for** $j = 1$ to m **do**
6: $N_{dC} = N_{dC} + diff(\tau(\cdot), R_i, I_j) \cdot d(i, j)$
7: **for** $F = 1$ to f **do**
8: $N_{dF}[F] = N_{dF}[F] + diff(F, R_i, I_j) \cdot d(i, j)$
9: $N_{dC\&dF}[F] + diff(\tau(\cdot), R_i, I_j) \cdot diff(F, R_i, I_j) \cdot d(i, j)$
10: **end for**
11: **end for**
12: **end for**
13: **for** $F = 1$ to f **do**
14: $W[F] = N_{dC\&dF}[F]/N_{dC} - (N_{dF}[F] - N_{dC\&dF}[F])/(m - N_{dC})$
15: **end for**

translate the HMC problem to a multi- class one, therefore ignoring the hierarchical aspect. Having in mind that the definitions of the HMC distances in Section 2 are actually weighted Euclidean, they are more suited to be included in the RReliefF algorithm, originally designed for regression.

The details of the RReliefF algorithm are given in pseudocode form in Algorithm 1. The algorithm begins by selecting a random instance (R_i) and finding the k nearest instances I_j to it. From these instances it then approximates the relevance $W[F]$ from Equation 4 of each feature in the following way.

First, we introduce the notation:

$$P_{diffC|diffF}(\text{diff. prediction}|\text{diff. value of F and nearest instances})$$

and then by using the Bayes rule, we have:

$$W[F] = \frac{P_{diffC|diffF} P_{diffF}}{P_{diffC}} - \frac{(1 - P_{diffC|diffF}) P_{diffF}}{1 - P_{diffC}} \tag{5}$$

The probabilities are estimated from N_{dC}, $N_{dF}[F]$ and $N_{dC\&dF}[F]$, where each of them is calculated as described in lines 6,8 and 9 from Algorithm 1. The estimations of these values is based on the distance calculation in the feature space, $diff(F, R_i, I_j)$, (lines 8 and 9) and in the target space, $diff(\tau(\cdot), R_i, I_j)$, (lines 6 and 9).

Our original purpose is to extend the RReliefF algorithm for hierarchical multi-label classification problems. Considering that the HMC refers to the target space, we extend the RReliefF algorithm by editing the way that $diff(\tau(\cdot), R_i, I_j)$ is calculated. From Section 2 and Equation 1 we obtain:

$$diff(\tau(\cdot), R_i, I_j) = diff(S_i, S_j) = \sqrt{\sum_k w(c_k)(v_{i,k} - v_{j,k})^2} \tag{6}$$

where S_i and S_j are the target descriptions of R_i and I_j correspondingly, while $v_{i,k}$ and $v_{j,k}$ are their binary representations. In this way, by changing the way the distance is calculated, the original RReliefF algorithm is extended to work for HMC problems. We name this extension HMC-ReliefF and we test its properties on various datasets in the following text.

4 Experiments

Our experimental evaluation of the HMC-ReliefF is based on the intuition of what is the expected output of a good feature ranking algorithm. Namely, a good feature ranking algorithm, would output the relevant features on top of the ranked list of features. A bad ranking algorithm, would not necessarily be the one that gives an inverse ranking according to relevance, but the one that outputs a random ranking. This means, that in the random ranking, the distribution of the relevant features is expected to be uniform throughout the list.

Having this in mind, we employ a stepwise filter-like procedure [18] to evaluate our HMC-ReliefF algorithm. The idea is that starting from the ranked list of features, we construct classifiers for different number of top-k ranked features. If there are relevant features on top of the feature ranking, then we can construct a classifier that has a good predictive performance. If the ranking is random then the number of relevant features in the top-k ranked features is expected to be smaller.

Formally, if we have a feature ranking algorithm r that we use on a dataset $\mathscr{D}$, then the output would be a feature ranking $\mathbf{R}$, namely:

$$r(\mathscr{D}) \rightarrow \mathbf{R}.$$

The feature ranking $\mathbf{R}$ is defined as an ordered list of features F, more specifically:

$$\mathbf{R} = (F_{r1}, \ldots, F_{rj}, \ldots, F_{rk})$$

where:

$$rank(F_{r1}) \leq \cdots \leq rank(F_{rj}) \leq \cdots \leq rank(F_{rk})$$

If we assume that we can induce and evaluate a predictive model $\mathscr{M}(R_i, F_t)$, where $R_i \subseteq \mathbf{R}$ and F_t is a target feature, then our whole evaluation procedure can be described as in Algorithm 2.

Algorithm 2 Stepwise evaluation of the top-k ranked features

Input: Feature Ranking, $\mathbf{R} = \{F_{r1}, \ldots, F_{rn}\}$; Target Feature, F_t
Output: FFA Curve, FFA, where $|FFA| = n$

```
R_S ⇐ ∅
for k = 1 to n do
    R_S ⇐ R_S ∪ feature(R, i)
    FFA[i] = qual(M(R_S, F_t))
end for
    return FFA
```

Table 1: Properties of the datasets with hierarchical targets; N_{tr} is the number of instances in the training dataset, D/C is the number of descriptive attributes (discrete/continuous), $|\mathscr{H}|$ is the number of classes in the hierarchy, $\mathscr{H}_d$ is the maximal depth of the classes in the hierarchy, $\overline{\mathscr{L}}$ is the average number of labels per example, and $\overline{\mathscr{L}_L}$ is the average number of leaf labels per example. Note that the values for $\mathscr{H}_d$ are not always a natural number because the hierarchy has a form of a DAG and the maximal depth of a node is calculated as the average of the depths of its parents.

Domain	N_{tr}	$\|D\|/\|C\|$	$\|\mathscr{H}\|$	$\mathscr{H}_d$	$\overline{\mathscr{L}}$	$\overline{\mathscr{L}_L}$
ImCLEF07D[5]	10000	0/80	46	3.0	3.0	1.0
ImCLEF07A[5]	10000	0/80	96	3.0	3.0	1.0
Reuters [14]	3000	0/47236	100	4.0	3.20	1.20
SCOP-GO [3]	6507	0/2003	523	5.5	6.26	0.95
SCOP-FUN [3]	2055	0/2003	250	4.0	3.42	0.95

For each step k of the filtering, we induce a classification model and evaluate its performance. This process of generating feature sets from the feature ranking is performed in a forward manner, by adding more and more of the top ranked features, which we name *forward feature addition* (FFA). At the end, we obtain a vector of model quality estimates that we can plot as a curve, thus obtaining a *FFA curve* that we use to estimate the performance of the feature ranking algorithm.

4.1 Experimental Setup

In this section, we give the details of the specific experimental setup that we consider.

From the description of the HMC-ReliefF algorithm, given in Algorithm 1, there are two basic parameters on which the produced results depend. Those are the number of random instances that are chosen m and the number of nearest neighbours k that are used to calculate the feature relevance values. Therefore, we decided to explore a reasonable set of values of these parameters in order to evaluate the algorithm performance. More specifically, for the number of random instances m and the number of nearest neighbours k, we consider the following parameters:

- $m = \{10, 50, 100, 250, 500\}$
- $k = \{10, 25, 50, 100\}$.

As a baseline for our comparisons we use a set of 50 random rankings for each different dataset. For each of these rankings we perform the previously described procedure in Section 4 and we generate a separate FFA curve. For the random rankings, we average the results of the 50 individual FFA curves, thus generating an expected FFA curve for a given dataset.

As a predictive model which we induce and evaluate, we use random forests of the so-called predictive clustering trees for hierarchical multi-label classification (PCT-HMCs)[19], [10]. The specific parameters that we used for the random forests of PCTs were 100 trees and a feature subset size of 10% of the dataset. In the HMC context,

there are various error measures that can be considered. We use the area of a variant of precision-recall curve, namely the Pooled Area Under the Precision-Recall Curve ($AU(\overline{PRC})$), details given in [19]. For estimating the $AU(\overline{PRC})$, we perform a ten-fold cross validation.

For the experiments we use datasets from various domains, which have classes organized in a hierarchy. We use 5 datasets from 3 domains, more specifically: biology (*SCOP-GO* and *SCOP-FUN*), text classification (*Reuters*) and image annotation/classification (*ImCLEF07D, ImCLEF07A*). The relevant parameters that characterize each dataset are given in Table 1. Note that the SCOP-GO dataset has a hierarchy organized as a DAG, while the remaining datasets have tree-shaped hierarchies. For more details on the datasets, we refer the reader to the referenced literature.

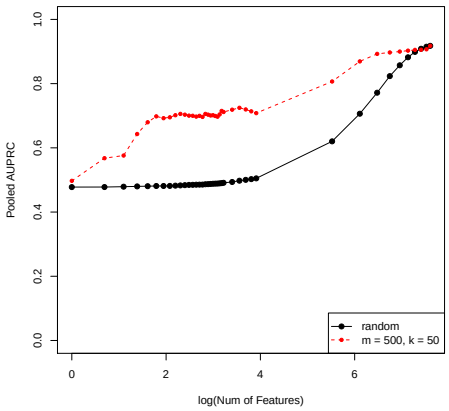
4.2 Results

In this section, we present the results from our experimental evaluation. In Figure 2, we give the results for the FFA curves for all of the datasets. For simplicity, we just compare the FFA curves for a single setting ($m = 500$, $k = 50$) and the expected FFA curves. An initial observation that can be made about all of the results is that the FFA curves of the HMC-ReliefF algorithm are most of the time above the FFA curves of the random rankings. This means that on top of the rankings produced by HMC-ReliefF, for different settings of m and k, relevant features can be found. It also means that this is not by chance, as the $AU(\overline{PRC})$ of the produced models is larger than the expected value of a random ranking.

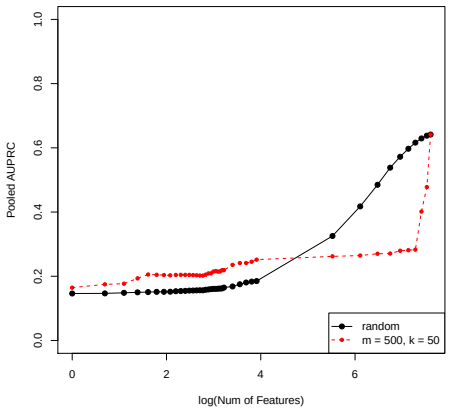
From the results, also some conclusions can be drawn about the behaviour of the HMC-ReliefF algorithm for different values of m and k. In Figure 3, we present a comparison of the FFA curves for a single dataset, namely *SCOP-GO*. The results are organised in four graphs, each showing FFA curves for a fixed k and different values for m, all compared to the expected FFA curve.

From all of the graphs, it can be observed that irrespectively of the number of neighbours k, a larger number of random instances m produces a better ranking. The best are obtained by setting the value of m to 250 and 500. This is also true for the *Reuters* dataset, while for the $ImCLEF07A$ and $ImCLEF07D$ datasets the results are not influenced by the change of the m and k parameters (results not shown).

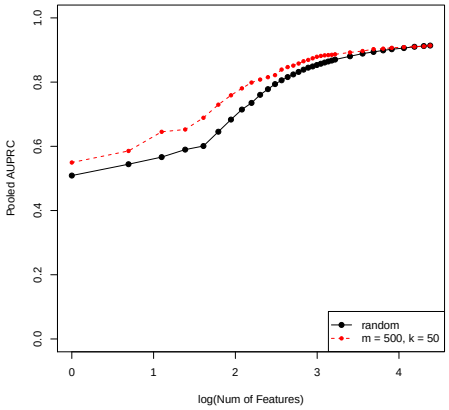
Opposite to the *SCOP-GO* dataset, the *SCOP-FUN* dataset seems to produce better results for a smaller k as well as a smaller m. This seems somewhat counter-intuitive considering that both of the datasets are from the same domain and share the same description space. However, we hypothesise that the different types of hierarchies considered (GO is a DAG and FUN is a tree) and their properties influenced these results.



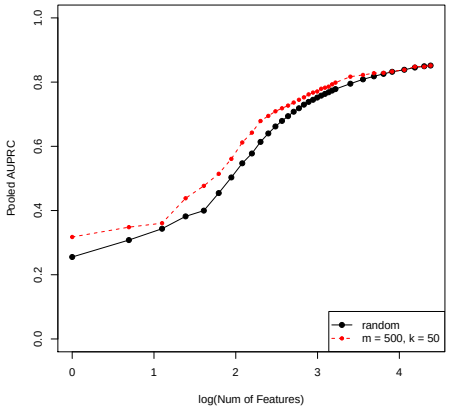
(a) FFA curves of **SCOP-GO**, **m = 500**, **k = 50**



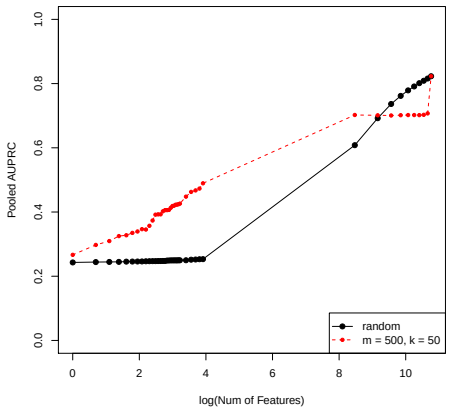
(b) FFA curves of **SCOP-FUN**, **m = 500**, **k = 50**



(c) FFA curves of **ImCLEF07D**, **m = 500**, **k = 50**

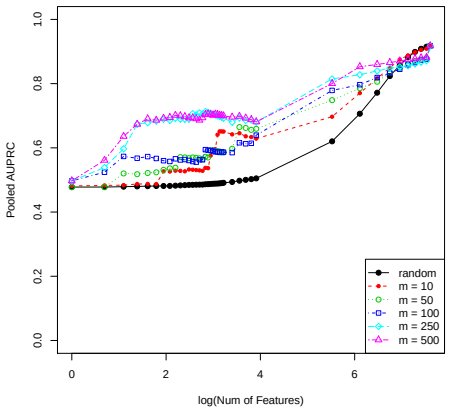


(d) FFA curves of **ImCLEF07A**, **m = 500**, **k = 50**

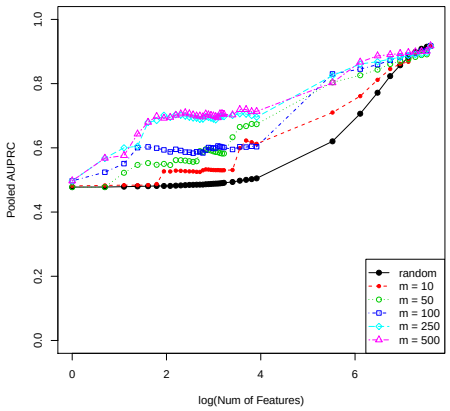


(e) FFA curves of **Reuters**, **m = 500**, **k = 50**

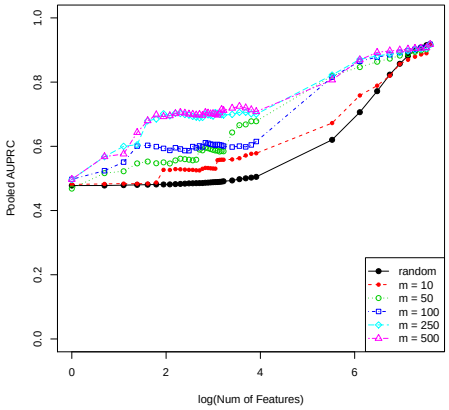
Fig. 2: Comparison of FFA curves of all datasets, for a fixed $m = 500$ and $k = 50$



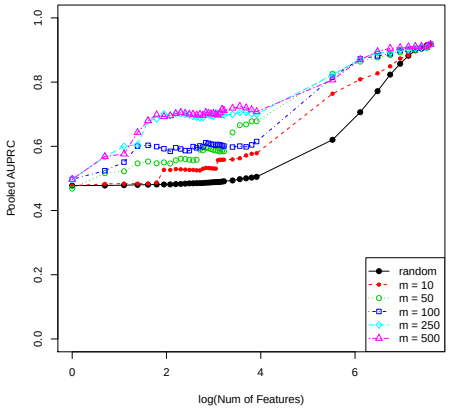
(a) FFA curves of **SCOP-GO**, **k = 10**



(b) FFA curves of **SCOP-GO**, **k = 25**



(c) FFA curves of **SCOP-GO**, **k = 50**



(d) FFA curves of **SCOP-GO**, **k = 100**

Fig. 3: Comparison of different FFA curves obtained by varying the number of *m* and *k* for the SCOP-GO dataset

5 Conclusions and Further Work

In this paper, we presented the HMC-ReliefF algorithm, which is an extension of the RReliefF algorithm for the task of Hierarchical Multi-label Classification. We believe that this is both an interesting and novel line of work, with regards to feature ranking algorithms. To the best of our knowledge, there has not been any work for feature

ranking within the context of structured data. We specifically focused on the ReliefF algorithm, due to its success in both classification and regression settings. The specific type of structured problems that we considered (HMC), was motivated by the fact that this kind of data can be found in various domains including biology, text mining and image annotation.

We evaluated the HMC-ReliefF algorithm on several datasets from different domains and with different properties of the hierarchies. We first investigated if our algorithm was able to detect relevant features in a dataset and put them on top of the ranking. This, we consider is a minimum requirement of any feature ranking algorithm. Additionally, we also explored a reasonable set of parameter settings of the HMC-ReliefF, which has influence on the feature relevance estimations.

The results of our experiments showed that for various datasets, the HMC-ReliefF algorithm performed well, as evaluated by a stepwise filter like approach of constructing FFA curves. This performance was compared to an expected FFA curve, obtained from a set of random rankings. The exploration of the various parameters of the HMC-ReliefF showed the following. For two of the datasets, a large value of m was preferred, irrespectively of the number of neighbours k. The FFA curves of the other two datasets (from the image annotation domain) remained stable for different values of the parameters. On the remaining dataset, the algorithm performed well for small values of m and k. These effects might have various causes, like the type and properties of the hierarchies considered, which at this time remain unexplored.

With this paper and the results presented we performed an initial investigation of the HMC-ReliefF algorithm. The directions of further work with respect to our HMC-ReliefF algorithm are numerous. One major direction of work, would be to define an artificial, controlled setting for investigating HMC problems. Different types of hierarchies should be considered, which are also differently structured (balanced vs. unbalanced, different width, different depth), or differently populated by instances (sparse vs. non-sparse). Within this setting, the effects of the various parameters of HMC-ReliefF can be investigated and also the advantages and limitations of the algorithm can be explored.

References

1. M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature genetics*, 25(1):25–29, May 2000.
2. Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271, 1997.
3. Amanda Clare. *Machine learning and data mining for yeast functional genomics*. PhD thesis, University of Wales Aberystwyth, Aberystwyth, Wales, UK, 2003.
4. Thomas G. Dietterich, Pedro Domingos, Lise Getoor, Stephen Muggleton, and Prasad Tadepalli. Structured machine learning: the next ten years. *Machine Learning*, 73(1):3–23, 2008.
5. Ivica Dimitrovski, Dragi Kocev, Suzana Loskovska, and Sašo Džeroski. Hierchical annotation of medical images. In *Proceedings of the 11th International Multiconference - Information Society IS 2008*, pages 174–181. IJS, Ljubljana, 2008.

6. Thomas Gärtner and Shankar Vembu. On structured output training: hard cases and an efficient alternative. *Machine Learning*, 76:227–242, 2009.
7. Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.
8. Zengyou He and Weichuan Yu. Review article: Stable feature selection for biomarker discovery. *Comput. Biol. Chem.*, 34:215–225, August 2010.
9. Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In *ML92: Proceedings of the ninth international workshop on Machine learning*, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.
10. Dragi Kocev, Celine Vens, Jan Struyf, and Sašo Džeroski. Tree ensembles for predicting structured outputs. *Pattern Recognition*, 46(3):817–833, 2013.
11. Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97:273–324, 1997.
12. Igor Kononenko. Estimating attributes: Analysis and extensions of relief. In *European Conference on Machine Learning*, pages 171–182, 1994.
13. Hans-Peter Kriegel, Karsten Borgwardt, Peer Kröger, Alexey Pryakhin, Matthias Schubert, and Arthur Zimek. Future trends in data mining. *Data Mining and Knowledge Discovery*, 15:87–97, 2007.
14. David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.
15. Marko Robnik-Šikonja and Igor Kononenko. An adaptation of relief for attribute estimation in regression. In Douglas H. Fisher, editor, *ICML*, pages 296–304. Morgan Kaufmann, 1997.
16. Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Mach. Learn.*, 53:23–69, October 2003.
17. Carlos Silla and Alex Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.
18. Ivica Slavkov. *An Evaluation Method for Feature Rankings*. PhD thesis, IPS Jožef Stefan, Ljubljana, Slovenia, 2012.
19. Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

XML Document Partitioning using Ensemble Clustering

Gianni Costa, Riccardo Ortale
{costa,ortale}@icar.cnr.it

ICAR-CNR,
I87036 Rende (CS), ITALY

Abstract. In this paper we propose a new technique for partitioning XML documents, in which conventional clustering techniques operating on flattened representations of individual aspects of the XML documents are combined to partition the available XML corpus. This offers the potential to divide the problem of catching content and structural regularities into simpler subproblems, in which only individual aspects of the XML documents (i.e., either their content, structure or, even, some simple form of nesting of the former into the latter) are taken into account in isolation to perform multiple clusterings of the same XML corpus, with the ultimate purpose of obtaining a refined partition. The results of a preliminary empirical evaluation reveal the effectiveness of our approach.

1 Introduction

The eXtensible Markup Language[1] (XML) [1] is a standard for representing, storing and sharing data encoded as text files in which pure content can be flexibly arranged into a convenient logical structure.

Partitioning XML documents is useful in several applicative settings, including query processing, data integration and indexing, information retrieval and extraction, Web mining, classification of XML documents into hierarchical topic directories for content management and bioinformatics.

In its most general form, the task of partitioning XML documents [4] consists in separating a collection of XML documents into disjoint subsets called clusters. However, from a knowledge discovery point of view, finding clusters in an XML corpus is problematic for various reasons [2, 4]. First, conventional partitioning methods cannot be applied directly to XML documents and, even their adaptation to the XML domain is not trivial. Also there is a challenging research issue, that in principle involves dealing with various aspects of XML documents, e.g., aligning and matching their (sub)structures, catching content overlap, uncovering mutual semantic relationships among respective textual content and/or structural labels along with accounting for the occurrences of common contents into similar (sub)structures. Simpler models for data representation (such as,

[1] http://www.w3c.org

e.g, the vector space model [18]) do not allow to effectively represent the occurrences of text content within structures, whereas more sophisticated models (such as, e.g, the tensor space model [6]) may be demanding in terms of space and processing time. Yet, the number of XML documents is likely to be very large. Additionally, the dimensionality of XML documents (especially those whose nature is of the text-centric type) is huge, since the hierarchical logical structure exacerbates the curse of dimensionality experienced in text clustering. Yet, evaluating content and structural similarities between volumes of XML documents with a huge dimensionality is likely to be detrimental for the effectiveness, efficiency as well as scalability of the partitioning process.

In this paper we propose an unexplored perspective on XML document partitioning, based on combining multiple clusterings [11, 13, 15, 16], in order to obtain a refined partition that satisfies as much as possible multiple clustering criteria. Clustering combination is an especially interesting method, that allows to tackle XML document partitioning from multiple viewpoints. Although the spectrum of possible combinations is very wide and the idea of combining dedicated clustering approaches is certainly interesting and worthy of a deeper investigation, we believe that the joint exploitation of conventional clustering techniques is more appealing, since this offers the potential to avoid the complexity of designing suitable clustering algorithms, that evaluate the similarity between XML documents by taking into account all of the foresaid aspects of XML resemblance simultaneously. We present an XML partitioning technique, XPEC (*XML Partitioning based on Ensemble Clustering*), whose main novelty is the instantiation of the ensemble clustering method through the original combination of three distinct clusterings, in which consolidated and recent developments from the fields of structural XML clustering, information retrieval as well as (high-dimensional) transactional clustering are used. The three clusterings are operated through conventional partitioning approaches, by separately dividing the underlying collection of XML documents with respect to its structure, content and combination of the former two aspects in isolation. The results of a preliminary comparative evaluation on standard benchmark XML corpora reveals the effectiveness of the devised technique.

2 Preliminaries

In this section we introduce the notation adopted throughout the paper along with some basic concepts.

2.1 Tree-based XML Document Representation

The structure and content of the generic XML document with no references [1] can be modeled in terms of a suitable XML tree representation, that refines the traditional notion of *rooted labeled tree* to also catch content and its nesting into structure.

*An **XML tree** is a rooted, labeled, tree, represented as a tuple* $\mathbf{t} = (\mathbf{V_t}, r_\mathbf{t}, \mathbf{E_t}, \lambda_\mathbf{t})$*, whose elements have the following meaning.* $\mathbf{V_t} \subseteq \mathbb{N}$ *is a set of nodes and* $r_\mathbf{t} \in \mathbf{V_t}$

is the root node of $\mathbf{t}$*, i.e. the only node with no entering edges.* $\mathbf{E_t} \subseteq \mathbf{V_t} \times \mathbf{V_t}$ *is a set of edges, catching the parent-child relationships between nodes of* $\mathbf{t}$*. Finally,* $\lambda_t : \mathbf{V_t} \mapsto \Sigma$ *is a node labeling function, where* Σ *is an alphabet of node tags (i.e., labels).*

Notice that the elements of XML documents are not distinguished from their attributes in an XML tree: both are mapped to nodes in the corresponding XML-tree representation.

Let $\mathbf{t}$ be a generic XML tree. Nodes in $\mathbf{V_t}$ can be divided into two disjoint subsets: the set $\mathbf{L_t}$ of *leaves* and the set $\mathbf{V_t} - \mathbf{L_t}$ of *inner nodes*. An inner node has at least one child. A leaf is instead a node with no children, that can contain only textual information.

A root-to-leaf path $p_l^{r_\mathbf{t}}$ in $\mathbf{t}$ is a sequence of nodes encountered along the path from the root $r_\mathbf{t}$ to a leaf node l in $\mathbf{L_t}$, i.e., $p_l^{r_\mathbf{t}} =< r_\mathbf{t}, \ldots, l >$. Notation $\lambda_\mathbf{t}(p_l^{r_\mathbf{t}})$ denotes the sequence of labels that are associated in the XML tree $\mathbf{t}$ to the nodes of path $p_l^{r_\mathbf{t}}$, i.e., $\lambda_\mathbf{t}(p_l^{r_\mathbf{t}}) =< \lambda_\mathbf{t}(r_\mathbf{t}), \ldots, \lambda_\mathbf{t}(l) >$. The set of all root-to-leaf paths in $\mathbf{t}$ is denoted as $paths(\mathbf{t}) = \{p_l^{r_\mathbf{t}} | l \in \mathbf{L_t}\}$.

Let l be a leaf in $\mathbf{L_t}$. The set $terms(l) = \{w_1, \ldots, w_h\}$ is a model of the content items provided by l. Elements w_i (with $i = 1 \ldots h$) are as many as the distinct terms in the context of l. Notation $\lambda_\mathbf{t}(p_l^{r_\mathbf{t}}).w_h$ indicates an enriched path and will be used to explicitly represent the nested occurrence of the content item w_h in the structural context of the labeled root-to-leaf path $p_l^{r_\mathbf{t}}$. The content of an XML tree $\mathbf{t}$ is denoted as $terms(\mathbf{t}) = \cup_{l \in \mathbf{L_t}} terms(l)$. The set of all enriched paths in $\mathbf{t}$ is instead indicated as $paths^{(e)}(\mathbf{t}) = \cup_{l \in \mathbf{L_t}, w \in terms(l)} \{\lambda_\mathbf{t}(p_l^{r_\mathbf{t}}).w\}$.

Hereinafter, the notions of XML documents and XML tree will be used interchangeably.

2.2 Problem Statement

Our approach combines multiple clusterings of a same XML corpus to eventually find a refined partition. Formally, let $\mathcal{D} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$ be a forest of N XML trees. Assume that $\mathbf{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_h\}$ is an ensemble of h separate clusterings of $\mathcal{D}$, that can in principle be performed by different algorithms operating on distinct feature sets. The generic $\mathcal{P}_k \in \mathbf{P}$ is a set of l_h clusters, i.e., $\mathcal{P}_k = \{C_1^{(k)}, \ldots, C_{l_h}^{(k)}\}$, such that $\mathcal{D} = \cup_{i=1}^{l_h} C_i^{(k)}$. Ensemble clustering is exploited to build one partition $\mathcal{P} = \{C_1, \ldots, C_l\}$ of l nonempty clusters from $\mathbf{P}$, such that any two XML trees within the same generic cluster $C_i \in \mathcal{P}$ share meaningful content and structural regularities, whereas any two XML trees belonging to as many distinct clusters $C_i, C_j \in \mathcal{P}$ exhibit no relevant similarities.

We develop a specific instance of the general ensemble-clustering method, in which $\mathbf{P}$ consists of three clusterings of $\mathcal{D}$, i.e., $h = 3$ and $\mathcal{P}$ is a partition of $\mathcal{D}$.

2.3 XML Features and Ensemble Clusterings

The tree-based representation of XML documents enables the identification of the sets of XML features addressed in the context of the foresaid three clusterings, which are respectively denoted as $\mathcal{S}^{(p)}$, $\mathcal{S}^{(w)}$ and $\mathcal{S}^{(pw)}$. $\mathcal{S}^{(p)}$ is set of all

distinct root-to-leaf paths in $\mathcal{D}$. $\mathcal{S}^{(w)}$ is a collection of relevant terms for the individual XML documents, i.e., $\mathcal{S}^{(w)} \subset \cup_{\mathbf{t} \in \mathcal{D}} terms(\mathbf{t})$. $\mathcal{S}^{(pw)}$ is a space of relevant enriched paths, i.e., $\mathcal{S}^{(pw)} \subset \cup_{\mathbf{t} \in \mathcal{D}} paths^{(e)}(\mathbf{t})$.

The availability of the sets of XML features $\mathcal{S}^{(p)}$, $\mathcal{S}^{(w)}$ and $\mathcal{S}^{(pw)}$ allows to project the XML trees of $\mathcal{D}$ into suitable corresponding spaces.

More precisely, a space $\mathcal{F}^{(p)}$ is associated with the set $\mathcal{S}^{(p)}$ of structural features, i.e., $\mathcal{F}^{(p)} \triangleq \{\mathcal{F}_{\mathbf{p}} | \mathbf{p} \in \mathcal{S}^{(p)}\}$. The generic element of $\mathcal{F}^{(p)}$ is a boolean attribute, whose value indicates the presence (or absence) of the corresponding root-to-leaf path $\mathbf{p}$ within the individual XML trees of $\mathcal{D}$. Hence, the occurrence of the individual root-to-leaf paths within each XML tree can be explicitly represented by modeling the XML trees as transactions. Assume that $\mathbf{x}^{(\mathbf{t},p)}$ is the transactional representation of an XML tree $\mathbf{t}$ over $\mathcal{F}^{(p)}$. The value of each attribute $\mathcal{F}_{\mathbf{p}}$ within $\mathbf{x}^{(\mathbf{t},p)}$ is true if $\mathbf{p}$ is a root-to-leaf path of $\mathbf{t}$, otherwise it is false. Therefore, $\mathbf{x}^{(\mathbf{t},p)}$ can be modeled as a proper subset of $\mathcal{F}^{(p)}$, namely $\mathbf{x}^{(\mathbf{t},p)} \triangleq \{\mathcal{F}_{\mathbf{p}} \in \mathcal{F}^{(p)} | \mathbf{p} \in paths(\mathbf{t})\}$, with the meaning that the structural features explicitly present in $\mathbf{x}^{(\mathbf{t},p)}$ take value true, whereas the others assume value false. The transactional representation of $\mathcal{D}$ is denoted as $\mathbf{D}^{(p)} = \{\mathbf{x}^{(\mathbf{t},p)} \subset \mathcal{F}^{(p)} | \mathbf{t} \in \mathcal{D}\}$. To deal with the peculiarities of the transactional setting, $\mathbf{D}^{(p)}$ is partitioned through a clustering scheme called Generate-Clusters, that was proposed in [7]. Generate-Clusters is an effective and parameter-free algorithm for transactional clustering, that scales to process volumes of high-dimensional transactions and automatically partitions them into a natural number of clusters. Generate-Clusters operates by progressively partitioning the available transactions, on the basis of the gain in quality of the resulting clusters with respect to the whole transactional database.

Information retrieval methods [3, 5] are instead adopted to represent the relevance of the features from $\mathcal{S}^{(w)}$ and $\mathcal{S}^{(pw)}$ in the context of the individual XML trees. In particular, the vector space model is chosen to represent the XML trees as vectors in the high-dimensional spaces constituted by the elements of either $\mathcal{S}^{(w)}$ or $\mathcal{S}^{(pw)}$ and the *TFIDF* weighting scheme [17] is employed to quantify feature relevance. Formally, let $\overrightarrow{x}^{(\mathbf{t},w)}$ and $\overrightarrow{x}^{(\mathbf{t},pw)}$ be the vector representations of the generic XML tree $\mathbf{t}$ in $\mathcal{D}$ over $\mathcal{S}^{(w)}$ and $\mathcal{S}^{(pw)}$, respectively. $\overrightarrow{x}^{(\mathbf{t},w)}$ and $\overrightarrow{x}^{(\mathbf{t},pw)}$ have as many entries as the elements of the corresponding feature sets $\mathcal{S}^{(w)}$ and $\mathcal{S}^{(pw)}$. The i-th entries of $\overrightarrow{x}^{(\mathbf{t},w)}$ and $\overrightarrow{x}^{(\mathbf{t},pw)}$, denoted as $\overrightarrow{x}_i^{(\mathbf{t},w)}$ and $\overrightarrow{x}_i^{(\mathbf{t},pw)}$, indicate the *TFIDF* value of the respective elements of $\mathcal{S}^{(w)}$ and $\mathcal{S}^{(pw)}$, namely $\mathcal{S}_i^{(w)}$ and $\mathcal{S}_i^{(pw)}$. Specifically, $\overrightarrow{x}_i^{(\mathbf{t},w)} = TFIDF(\mathbf{t}, \mathcal{S}_i^{(w)})$ and $\overrightarrow{x}_i^{(\mathbf{t},pw)} = TFIDF(\mathbf{t}, \mathcal{S}_i^{(pw)})$. The *TFIDF* value of the generic element of $\mathcal{S}^{(w)}$ or $\mathcal{S}^{(pw)}$ is defined as the product of two factors, i.e., $TFIDF(\mathbf{t}, \mathcal{S}_i^{(\bullet)}) = TF(\mathbf{t}, \mathcal{S}_i^{(\bullet)}) \cdot IDF(\mathbf{t}, \mathcal{S}_i^{(\bullet)})$. In turn, $TF(\mathbf{t}, \mathcal{S}_i^{(\bullet)})$ is the occurrence frequency of $\mathcal{S}_i^{(\bullet)}$ in $\mathbf{t}$. $IDF(\mathbf{t}, \mathcal{S}_i^{(\bullet)})$ indicates the inverse document frequency of $\mathcal{S}_i^{(\bullet)}$ in $\mathcal{D}$, i.e., $IDF(\mathbf{t}, \mathcal{S}_i^{(\bullet)}) = log \frac{1+|\mathcal{D}|}{|\mathcal{D}_i|}$, where $\mathcal{D}_i$ is a subset of $\mathcal{D}$, such that for each XML tree $\mathbf{t}' \in \mathcal{D}_i$ it holds that $\mathcal{S}_i^{(\bullet)} \in terms(\mathbf{t}')$ (if $\bullet$ is a placeholder for w) or $\mathcal{S}_i^{(\bullet)} \in paths^{(e)}(\mathbf{t}')$ (if $\bullet$ is a placeholder for pw). Notation $\mathbf{D}^{(w)}$ and $\mathbf{D}^{(pw)}$

is used to indicate the vector representations of $\mathcal{D}$, i.e., $\mathbf{D}^{(w)} = \{\overrightarrow{x}^{(\mathbf{t},w)} | \mathbf{t} \in \mathcal{D}\}$ and $\mathbf{D}^{(pw)} = \{\overrightarrow{x}^{(\mathbf{t},pw)} | \mathbf{t} \in \mathcal{D}\}$.

A globally-optimized repeated-bisecting method, equipped with the cosine distance, is leveraged to partition both $\mathbf{D}^{(w)}$ and $\mathbf{D}^{(pw)}$. In particular, we exploited the implementation available in the *Cluto Toolkit* [12] of the foresaid repeated-bisecting method, which will be henceforth referred to as RBR.

3 XML Partitioning based on Ensemble Clustering

The pseudo code of the XPEC approach is shown in Fig. 1. It works by projecting the individual XML trees of the input forest $\mathcal{D}$ into three distinct spaces of clustering features, separately partitioning the resulting representations and then combining such clusterings into one partition. More precisely, four different stages can be identified. At the first stage (lines 1-4), the XML trees are projected (at line 2) into a space of structural features, which results (at line 3) in the transactional representation $\mathbf{D}^{(p)}$ of the original forest $\mathcal{D}$. GENERATECLUSTERS is subsequently used (at line 4) to group the transactions within $\mathbf{D}^{(p)}$ in the partition $\mathcal{P}^{(p)}$. At the second stage (lines 5-8), the vector representation $\mathbf{D}^{(w)}$ of the original forest $\mathcal{D}$ results (at line 7) from the projection (at line 6) of the XML trees into a space of content items. The partition $\mathcal{P}^{(w)}$ is discovered (at line 8) by RBR in $\mathbf{D}^{(w)}$. At the third stage (lines 9-12), the XML trees are projected (at line 10) into a space of enriched paths to yield the vector representation $\mathbf{D}^{(pw)}$ (at line 11). The latter is partitioned by RBR (at line 12) to form the partition $\mathcal{P}^{(pw)}$. It is worth to underline that RBR requires the number of clusters to find as an input parameter. This is not specified in Fig. 1 for the sake of readability. However, in the experimental evaluation, the number of clusters discovered in both $\mathcal{P}^{(w)}$ and $\mathcal{P}^{(pw)}$ is reported in Fig. 1. Finally, at the forth stage (line 13), $\mathcal{P}^{(p)}$, $\mathcal{P}^{(w)}$ and $\mathcal{P}^{(pw)}$ are suitably combined to find one refined partition of $\mathcal{D}$. The COMBINE procedure used at line 13 of Fig. 1 is detailed in Fig. 2. Essentially, it can be explained as another round of transactional clustering in the space of boolean features induced by the input clusterings $\mathcal{P}_1, \ldots, \mathcal{P}_h$, according to the scheme for the combination of multiple clusterings presented in [15].

Further details on the clustering features used in the former three stages of XPEC are provided below.

3.1 Clustering Features

XPEC relies on the XCPC approach [9] to manipulate the XML documents by their structure alone. This is obtained (at line 1 of Fig. 1) by defining the set $\mathcal{S}^{(p)}$ of relevant structural features as the collection of all distinct root-to-leaf paths in $\mathcal{D}$, i.e., $\mathcal{S}^{(p)} = \cup_{\mathbf{t} \in \mathcal{D}} paths(\mathbf{t})$.

The set $\cup_{\mathbf{t} \in \mathcal{D}} terms(\mathbf{t})$ is the whole space of content items from $\mathcal{D}$. We resort to feature selection in order to distill a subset $\mathcal{S}^{(w)}$ of relevant content items from $\cup_{\mathbf{t} \in \mathcal{D}} terms(\mathbf{t})$, that maintain their original semantic meaning (as opposed to feature extraction methods, that instead would identify artificial features with

XPEC($\mathcal{D}$)
Input: a forest $\mathcal{D} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$ of XML trees;
Output: a partition $\mathcal{P}$ of $\mathcal{D}$;
1: let $\mathcal{S}^{(p)}$ be a set of relevant structural features in $\mathcal{D}$;
2: let $\mathbf{x}^{(\mathbf{t}_i,p)} \leftarrow \{\mathcal{F}_{\mathbf{p}} \in \mathcal{F}^{(p)} | \mathbf{p} \in \mathcal{S}^{(p)}\}$ for each $i = 1, \ldots, N$;
3: let $\mathbf{D}^{(p)} \leftarrow \{\mathbf{x}^{(\mathbf{t},p)} \subseteq \mathcal{F}^{(p)} | \mathbf{t} \in \mathcal{D}\}$;
4: $\mathcal{P}^{(p)} \leftarrow$ GENERATE-CLUSTERS($\mathbf{D}^{(p)}$);
5: let $\mathcal{S}^{(w)}$ be a set of representative content items from $\mathcal{D}$;
6: let $\overrightarrow{x}_i^{(\mathbf{t},w)} \leftarrow TFIDF(\mathbf{t}, \mathcal{S}_i^{(w)})$ for each $\mathbf{t} \in \mathcal{D}$ and $i = 1, \ldots, |\mathcal{S}^{(w)}|$;
7: let $\mathbf{D}^{(w)} \leftarrow \{\overrightarrow{x}^{(\mathbf{t},w)} | \mathbf{t} \in \mathcal{D}\}$;
8: $\mathcal{P}^{(w)} \leftarrow$ RBR($\mathbf{D}^{(w)}$);
9: let $\mathcal{S}^{(pw)}$ be a set of representative enriched paths from $\mathcal{D}$;
10: let $\overrightarrow{x}_i^{(\mathbf{t},pw)} \leftarrow TFIDF(\mathbf{t}, \mathcal{S}_i^{(pw)})$ for each $\mathbf{t} \in \mathcal{D}$ and $i = 1, \ldots, |\mathcal{S}^{(pw)}|$;
11: let $\mathbf{D}^{(pw)} \leftarrow \{\mathbf{x}^{(\mathbf{t},pw)} | \mathbf{t} \in \mathcal{D}\}$;
12: $\mathcal{P}^{(pw)} \leftarrow$ RBR($\mathbf{D}^{(pw)}$);
13: $\mathcal{P} \leftarrow$ COMBINE($\mathcal{P}^{(p)}, \mathcal{P}^{(w)}, \mathcal{P}^{(pw)}$);
14: RETURN $\mathcal{P}$;

Fig. 1. The scheme of the XPEC algorithm

COMBINE($\mathcal{P}_1, \ldots, \mathcal{P}_h$)
Input: multiple separate clusterings $\mathcal{P}_1, \ldots, \mathcal{P}_h$ of $\mathcal{D}$;
L1: /* form one partition $\mathcal{P}$ of $\mathcal{D}$, by combining $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(h)}$ through the scheme in [15], which is sketched by the below lines;
L2: $\mathcal{S} \leftarrow \{\mathcal{F}_{C_m^{(l)}} | l = 1, \ldots, h \wedge m = 1, \ldots, |\mathcal{P}^{(l)}|\}$;
L3: $\mathbf{x}^{(\mathbf{t}_i)} \leftarrow \{\mathcal{F}_{C_m^{(l)}} \in \mathcal{S} | \mathbf{t}_i \in C_m^{(l)}\}$ for each $i = 1, \ldots, N$;
L4: $\mathbf{D} \leftarrow \{\mathbf{x}^{(\mathbf{t}_i)} \subseteq \mathcal{S} | \mathbf{t}_i \in \mathcal{D}\}$;
L5: $\mathcal{P} \leftarrow$ GENERATE-CLUSTERS($\mathbf{D}$);
L6: RETURN $\mathcal{P}$;

Fig. 2. The ensemble clustering procedure used at line 13 of Fig. 1

less clear meaning). More precisely, the mean *TFIDF* value [19] of each content item over all the XML documents within $\mathcal{D}$ is used to quantify its relevance to the clustering process.

The task of clustering by both content and structure is functional to drive the combination of the former two clusterings, respectively performed with respect to content and structure in isolation. A simple form of content nesting into structure, namely the enriched paths, is targeted in this clustering process. Focusing on extended paths exacerbates the curse of dimensionality faced when dealing with content only. Therefore, the whole set of enriched paths $\cup_{\mathbf{t} \in \mathcal{D}} paths^{(e)}(\mathbf{t})$ is distilled (at line 9 of Fig. 1) into a smaller set $\mathcal{S}^{(pw)}$. Again, this is accomplished by exploiting the average *TFIDF* value [19] of the enriched paths. $\mathcal{S}^{(pw)}$ is then grown through the addition of constrained rarely occurring enriched paths. Details about the constraints on the occurrence frequency of the added rare enriched paths are provided in Section 4.

4 Evaluation

A preliminary empirical evaluation was conducted to assess the behavior of XPEC. The effectiveness of XPEC is comparatively evaluated against an established competitor. We also expect to gain some preliminary insight into the more general question of the actual advantages deriving from approaching the problem of XML partitioning through the ensemble clustering method. All tests are performed on a Linux machine, with an Intel Core 2 Duo cpu, 4Gb of memory and 2Ghz of clock speed.

The *Sigmod* corpus[2] was chosen as benchmark dataset. It consists of 140 XML documents complying to two different structural class DTDs, namely `IndexTermsPage` and `OrdinaryIssuePage`. Originally, this corpus was used to measure the effectiveness of the approaches to XML clustering at grouping XML documents by structure alone (e.g., in) [2, 8]. However, given the reduced number of structural classes, this is not a challenging task. Therefore, in the following tests, we consider a classification of the *Sigmod* corpus into 5 classes [14]. These were obtained for the purpose of adding complexity to the experimental evaluation, by using expert knowledge to group the underlying XML documents on the basis of their structure and content-based similarity.

On *Sigmod*, the effectiveness of XPEC is compared against the effectiveness of state-of-the-art competitor XCFS [14], whose performances on the chosen XML corpus are reported from [14]. In addition, we report the effectiveness of the clusterings combined by XPEC, for the twofold purpose of providing additional baselines and understanding whether their combination is actually an improvement with respect to the individual clusterings. These are respectively denoted as SO (which stands for the clustering of XML documents based on their structure alone, as it is obtained by applying the Generate-Clusters [7] procedure to the structural features selected in Section 3.1, that is equivalent to the approach in [9]), CO (which stands for the clustering of XML documents based on their content alone) and SC (which stands for the clustering of XML documents with respect to the enriched paths).

Effectiveness is measured in terms of *purity*. The latter measure was used in the context of the Mining Track at INEX 2007 [10] and is widely exploited in the literature. Purity is an external quality measure, that assumes knowledge of a predefined classification of the XML documents into a certain number k of natural classes. Therefore, we study the effectiveness of XPEC over collections of XML documents with known class labels and analyze the correspondence between the discovered and natural structures. The available class labels are hidden to XPEC. A partition $\mathcal{P} = \{C_1, \ldots, C_l\}$ of $\mathcal{D}$ can be summarized into a contingency table m, where columns represent discovered clusters and rows represent natural classes. Each entry m_{ij} indicates the number of XML documents in $\mathcal{D}$, that are assigned to cluster C_j and actually belong to the natural class $\mathbf{C}_i$, with $1 \leq i \leq k$. Intuitively, each cluster C_j corresponds to the class $\mathbf{C}_i$ that is best represented in C_j, i.e., such that m_{ij} is maximal. For any cluster C_j, the index $h(j)$ of the class with maximal m_{ij} is defined as $h(j) = max_i\ m_{ij}$. Purity for

[2] kindly provided by Professor Richi Nayak (personal communication)

a cluster C_j is a measure of the degree to which C_j contains XML documents primarily from $\mathbf{C}_{h(j)}$ [4]. Formally, $Purity(C_j) = \frac{|\mathbf{C}_{h(j)}|}{|C_j|}$. Macro-averaged purity and micro-averaged purity extend the notion of purity for a single cluster to a whole partition $\mathcal{P}$. Precisely, macro-averaged purity for partition $\mathcal{P}$ is defined as

$$Macro\text{-}averaged\ purity(\mathcal{P}) = \frac{1}{h} \sum_{C \in \mathcal{P}} Purity(C)$$

Macro-averaged purity is an arithmetic mean, that assigns a same weight to each cluster. Instead, micro-averaged purity weights each cluster by a weight proportional to the size of the cluster itself, i.e.,

$$Micro\text{-}averaged\ purity(\mathcal{P}) = \frac{\sum_{C \in \mathcal{P}} |C| \cdot Purity(C)}{N}$$

Obviously, micro-averaged purity is more strongly influenced by larger clusters. Both micro-averaged purity and macro-averaged purity are measured in our experiments. Larger values of such measures are indicative of higher partitioning effectiveness.

Table 1 shows the results of a comparative evaluation of the effectiveness of XPEC at partitioning XML documents. The empirical evidence reveals that XPEC is the best performer on *Sigmod* corpus, whose XML documents exhibit differentiated structures. Also the combination of SO, CO and SC results into an actual improvement of performance with respect to the individual clusterings.

Corpus	Competitor	Clusters	Micro-averaged purity	Macro-averaged purity
Sigmod	XPEC	5	0.83	0.84
	XCFS [14]	5	0.82	0.49
	SO (or, equivalently, [9])	2	0.79	0.80
	CO	5	0.80	0.83
	SC	5	0.80	0.82

Table 1. Comparative evaluation of clustering effectiveness

5 Conclusions and Further Research

We proposed to exploit the combination of multiple clustering for partitioning XML documents, since in principle it allows to decompose the inherently difficult problem of catching structural and content relationships into a number of simpler subproblems. To verify the validity of our intuition, we developed a technique, XPEC, which separately clusters a same corpus of XML documents with respect to their contents, structure and simple nesting of content items into root-to-leaf paths. The three clusterings are combined by solving a categorical clustering problem, as discussed in [15]. The results of a preliminary evaluation over a real-world XML corpus reveal the effectiveness of our approach. As further research, currently, we are exploring alternative patterns of content nesting into structure.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.
2. C.C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki. Xproj: A framework for projected structural clustering of xml documents. In *Proc. of Int. Conf. on Knowledge Discovey and Data Mining*, pages 46 – 55, 2007.
3. C.C.. Aggarwal and C. Zhai, editors. *Mining Text Data*. Springer, 2012.
4. A. Algergawy, M. Mesiti, R. Nayak, and G. Saake. Xml data clustering: An overview. *ACM Computing Surveys*, 43(4):25:1 – 25:41, 2011.
5. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
6. D. Cai, X. He, and J. Han. Tensor space model for document analysis. Technical report UIUCDCS-R-2006-2715, Computer Science Department, University of Illinois at Urbana-Champaign, 2006.
7. E. Cesario, G. Manco, and R. Ortale. Top-down parameter-free clustering of high-dimensional categorical data. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1607 – 1624, 2007.
8. G. Costa, G. Manco, R. Ortale, and E. Ritacco. Hierarchical clustering of xml documents focused on structural components. *Data and Knowledge Engineering*, 84:26 – 46, 2013.
9. G. Costa and R. Ortale. On effective xml clustering by path commonality: An efficient and scalable algorithm. In *IEEE Int. Conf. on Tools with Artificial Intelligence*, pages 389 – 396, 2012.
10. L. Denoyer and P. Gallinari. Report on the xml mining track at inex 2007. *ACM SIGIR Forum*, 42(1):22 – 28, 2008.
11. A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1):341 – 352, 2007.
12. G. Karypis. Cluto - software for clustering high-dimensional datasets. Available at `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download`.
13. L.I. Kuncheva, S.T. Hadjitoorov, and L.P. Todorova. Experimental comparison of cluster ensemble methods. In *Int. Conf. on Information Fusion*, pages 1 – 7, 2006.
14. S. Kutty, R. Nayak, and Y. Li. Xcfs: A novel approach for clustering xml documents using both the structure and the content. In *Proc. of ACM Conference on Information and Knowledge Management*, pages 1729 – 1732, 2009.
15. T. Li, M. Ogihara, and S. Ma. On combining multiple clusterings: an overview and a new perspective. *Applied Intelligence*, 33(2):207 – 219, 2010.
16. S. Vega Pons and J. Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(3):337 – 372, 2011.
17. G. Salton. Developments in automatic text retrieval. *Science*, 253:974 – 980, 1991.
18. G. Salton, A. Wong, and C.S. Yang. A vector space model for information retrieval. *Communications of the ACM*, 18:613620, 1975.
19. B. Tang, M. Shepherd, E. Milios, and M.I. Heywood. Comparing and combining dimension reduction techniques for efficient text clustering. In *Canadian Conference on AI*, pages 292–296, 2005.

Process Mining to Forecast the Future of Running Cases

Annalisa Appice[1], Sonja Pravilovic[1,2], and Donato Malerba[1]

[1]Dipartimento di Informatica, Università degli Studi di Bari Aldo Moro
via Orabona, 4 - 70126 Bari - Italy
[2]Montenegro Business School, Mediterranean University Vaka Djurovica
b.b. Podgorica - Montenegro
{annalisa.appice, sonja.pravilovic, donato.malerba}@uniba.it

Abstract. Processes are everywhere in our daily lives. More and more information about executions of processes are recorded in event logs by several information systems. Process mining techniques are used to analyze historic information hidden in event logs and to provide surprising insights for managers, system developers, auditors, and end users. While existing process mining techniques mainly analyze full process instances (cases), this paper extends the analysis to running cases, which have not yet completed. For running cases, process mining can be used to notify future events. This forecasting ability can provide insights for check conformance and support decision making. This paper details a process mining approach, which uses predictive clustering to equip an execution scenario with a prediction model. This model accounts for recent events of running cases to predict the properties of future events. Several tests with benchmark logs investigate the viability of the proposed approach.

1 Introduction

Contemporary systems, ranging from high-tech and medical devices to enterprise information systems and e-business infrastructures, record massive amounts of events by making processes visible. Each event has a name and additional mandatory properties which include the timestamp (i.e. exact date and time of occurrence), the lifecycle transition state (i.e. whether the event refers to a task having been started, completed) and the resource (i.e. name of the originator having triggered the event). In addition, each event may be characterized by several optional properties, such as cost, location, outcome, which are specific for the process. Process mining techniques [6] can be used to analyze event logs in order to extract, modify and extend process models, as well as to check conformance with respect to defined process models.

Thus far, several process mining techniques have been used in the discovery, conformance and enhancement of a variety of business processes [7]. They are mainly used in an off-line fashion and rarely for operational decision support. Historical full cases (i.e. instances of the process which have already completed) are analyzed, while running cases (i.e. instances of the process which have not

completed yet) are rarely processed on-line. However, a new perspective has emerged recently. van der Aalst et al [8] demonstrate that process mining techniques are not necessarily limited to the past, but can also be used for the present and the future. To make this philosophy concrete, they have already presented a set of approaches, which can be used very well for operational decision support. In particular, they propose to mine predictive process models from historic data and use them to estimate the remaining processing time and the probability of a particular outcome for running cases [9].

Embracing this philosophy, we consider another predictive task and detail a process mining approach to forecast future events of running cases. This forecasting service can be used to check conformance and recommend appropriate actions. In particular, we can check whether the observed event fits the forecast behavior. The moment the case deviates, an appropriate actor can be alerted in real time. Similarly, we can use forecasts to notify recommendations proposing/describing activity elements (e.g. resource, activity name, cost), which will comply the process model.

In this paper, we transform the task of event forecasting for running cases into a predictive clustering task [2], where the target variables are the properties of future events expected in running cases, while the predictors are properties of recent events up to a certain time window. This transformation technique is usually known as "time delay embedding" [5] and frequently used in stream data mining, where it is also called sliding window model [3]. Historical cases, transformed with the sliding window model, can be processed off-line so that a predictive clustering tree (PCT) [2] can be mined for the predictive aim. A PCT is a tree structured model, which generalizes decision tree by predicting many labels of an examples at once. In this study, it allows us to reveal in advance properties of future events based on properties of recent time-delayed event elements. The PCT can be used to forecast on-line event elements of any new running case. We have implemented this approach in the context of the ProM [10] framework, and verified its effectiveness in various case studies.

This paper is organized as follows. Section 2 introduces some notations and briefly revises the predictive clustering technique as well as the sliding window model used. Section 3 describes the event-based forecasting approach proposed. Section 4 demonstrates the usefulness of our approach using several benchmark case studies. Section 5 concludes the paper.

2 Basics

In this section, we introduce some basic concepts needed for the event forecasting. We describe event logs, as well as introduce the ideas behind predictive clustering and sliding window model.

2.1 Event log

The basic assumption is that the log contains information about activities executed for specific cases of a certain process type, as well as their durations.

Table 1. A fragment of the event log *repair* [7]. The symbol (*) identifies the optional properties which are specific for the process.

id	name	timestamp	resource	lifecycle	phone type*	defect type*	defect fixed*	number repairs*
1	Register	1970-01-02T12:23	System	complete	-	-	-	-
1	Analyze Defect	1970-01-02T12:23	Tester3	start	-	-	-	-
1	Analyze Defect	1970-01-02T12:30	Tester3	complete	T2	6	-	-
1	Repair (Complex)	1970-01-02T12:31	SolverC1	start	-	-	-	-
1	Repair (Complex)	1970-01-02T12:49	SolverC1	complete	-	-	-	-
1	Test Repair	1970-01-02T12:49	Tester3	start	-	-	-	-
1	Test Repair	1970-01-02T12:55	Tester3	complete	-	-	-	-
1	Inform User	1970-01-02T13:10	System	complete	-	-	-	-
1	Archive Repair	1970-01-02T13:10	System	complete	-	-	true	0
2	Register	1970-01-01T11:09	System	complete	-	-	-	-
2	Analyze Defect	1970-01-01T11:09	Tester2	start	-	-	-	-
2	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Definition 1 (Event, Property). *Let $\mathcal{E}$ be the event domain for a process $\mathcal{P}$. An event ϵ ($\epsilon \in \mathcal{E}$) is characterized by a set of* mandatory properties*, that is, the event corresponds to an activity, has a timestamp which represents date and time of occurrence, is triggered by a resource, refers to a specific lifecycle transition state which can be started or completed. Additionally, it can be characterized by variable* process-specific optional properties *such as cost, location, outcome and so on. Optional properties may also be away in an event.*

An event log is a set of events. Each event in the log is linked to a particular trace and globally unique. A trace in a log represents a process instance (e.g. a customer order or an insurance claim) also referred to as case. Time is non-decreasing in each case in the log.

Definition 2. *A case $\mathcal{C}$ is a finite sequence of events $e \in \mathcal{E}$ such that each event occurs only once (i.e. for $1 \leq i < j \leq |\mathcal{C}|$: $\epsilon(i) \neq \epsilon(j)$) and time is non-decreasing (i.e. $time(\epsilon(i)) \leq time(\epsilon(j))$. A log $\mathcal{L}$ is a bag of cases.*

A fragment of event log is reported in Table 1. A case containing nine events is shown. Each event has the mandatory properties and several optional properties. We note that the value may also lack in an event for an optional property.

2.2 Predictive clustering trees

The task of mining predictive clustering trees is now formalized. *Given*

- a descriptive space $\mathbf{X} = \{X_1, X_2, \ldots X_m\}$ spanned by m independent (or predictor) variables X_j,
- a target space $\mathbf{Y} = \{Y_1, Y_2, \ldots, Y_q\}$ spanned by q dependent (or target) variables Y_j,

– a set $\mathcal{T}$ of training examples, $(\mathbf{x_i}, \mathbf{y_i})$ with $\mathbf{x_i} \in \mathbf{X}$ and $\mathbf{y}_i \in \mathbf{Y}$

Find a tree structure τ which represents:

– A set of hierarchically organized clusters on T such that for each $u \in T$, a sequence of clusters $C_1, C_2, \ldots, C_k$ exist for which $u \in C_{i_r}$ and the containment relation $C_1 \supseteq C_2 \supseteq \ldots \supseteq C_k$ is satisfied. Clusters $C_1, C_2, \ldots, C_k$ are associated to the nodes $t_1, t_2, \ldots, t_k$, respectively, where each $t_i \in \tau$ is a direct child of $t_{i-1} \in \tau$ $(j = 1, \ldots, k)$, t_1 is the root of the structure τ and t_k is a leaf of the structure.
– A predictive piecewise function $f : \mathbf{X} \rightarrow \mathbf{Y}$, defined according to the hierarchically organized clusters. In particular,

$$\forall u \in \mathbf{X},\ f(u) = \sum_{t_i \in leaves(\tau)} D(u, t_i) f_{t_i}(u) \tag{1}$$

where $D(u, t_i) = \begin{cases} 1 \text{ if } u \in C_i \\ 0 \text{ otherwise} \end{cases}$ and $f_{t_i}(u)$ is a (multi-target) prediction function associated to the leaf t_i.

Clusters are identified according to both the descriptive space and the target space $\mathbf{X} \times \mathbf{Y}$. This is different from what is commonly done in predictive modeling and classical clustering, where only one of the spaces is typically considered. This general formulation of the problem allows us to have the prediction model mining phase which can consider multiple target variables at once. This is the case of predicting the "several" properties of the next event in a case.

The construction of a PCT is not very different from the construction of standard decision tree (see, for example, the C4.5 algorithm [4]): at each internal node t, a test has to be selected according to a given evaluation function. The main difference is that for a PCT, the best test is selected by maximizing the (inter-cluster) variance reduction over the target space $\mathbf{Y}$, defined as:

$$\Delta_Y(C, \mathcal{P}) = Var_{\mathbf{Y}}(C) - \sum_{C_i \in \mathcal{P}} \frac{|C_i|}{|C|} Var_{\mathbf{Y}}(C_i), \tag{2}$$

where C is the cluster associated with t and $\mathcal{P}$ defines the partition $\{C_1, C_2\}$ of C. The partition is defined according to a Boolean test on a predictor variable of the descriptive space $\mathbf{X}$. By maximizing the variance reduction, the cluster homogeneity is maximized, improving at the same time the predictive performance. $Var_{\mathbf{Y}}(C)$ is the variance of $\mathbf{Y}$ in the cluster C. It is computed as the average of variances of target variables $Y_j \in \mathbf{Y}$, that is, $Var_{\mathbf{Y}}(C) = \sum_{Y_j \in \mathbf{Y}} var_{Y_j}(C)$.

2.3 Sliding window model

A sliding window model is the simplest model to consider the recent data of a running case and run queries over the data of the recent past only. Originally

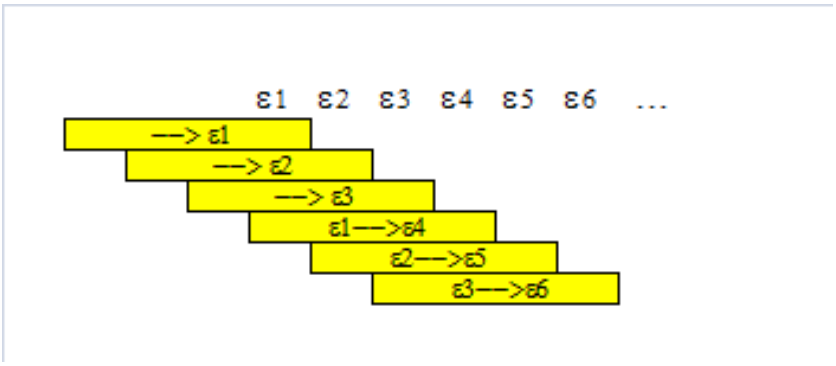


Fig. 1. Sliding window model of a running case with window size $w = 4$.

defined for data stream mining, this type of window is similar to the first-in, first-out data structure. When the event ϵ_i is acquired and inserted in the window, the latest event ϵ_{i-w} is discarded (see Figure 1). w is the size of the window.

Definition 3 (Sliding window model). *Let w be the window size of the model. A sliding window model views a case $\mathcal{C}$ as a sequence of overlapping windows of events,*

$$\mathcal{C}(1 \to w),\ \mathcal{C}(2 \to w+1),\ \ldots,\ \mathcal{C}(i-w+1 \to i),\ \ldots, \tag{3}$$

where $\mathcal{C}(i-w+1 \to i)$ is the series of the w events ϵ_{i-w+1}, ϵ_{i-w+2}, ..., ϵ_{i-1}, ϵ_i of the case $\mathcal{C}$ with $time(\epsilon_{i-j-1}) \leq time(\epsilon_{i-j})$ (for all $j = 0, \ldots, w-2$).

By considering the sliding window model, the window $\mathcal{C}(i-w+1 \to i)$ defines the recent history of the event ϵ_i.

3 Framework for PCT-based Event Forecasting

We use the sliding window model to transform our event-based forecasting problem into a predictive clustering problem where the target variables are the properties of the next event in the case. A PCT is learned off-line from an event log which records full cases and used on-line to forecast the next event expected in a running case (see Figure 2).

Let $\mathcal{L}$ be the event log which records full cases of a process $\mathcal{P}$. In the off-line phase, $\mathcal{L}$ is transformed in a training set $\mathcal{T}$. This transformation is performed by using the sliding window model. Let w be the window size. Each training case $\mathcal{C} \in \mathcal{L}$ is transformed in a bag $training(\mathcal{C}, w)$ of training examples. This bag collects a training example for each event ϵ_i of $\mathcal{C}$ so that the training example $\mathbf{xy}(\mathcal{C}(i-w+1 \to i))$ is generated based upon the sliding window history of the event. Formally,

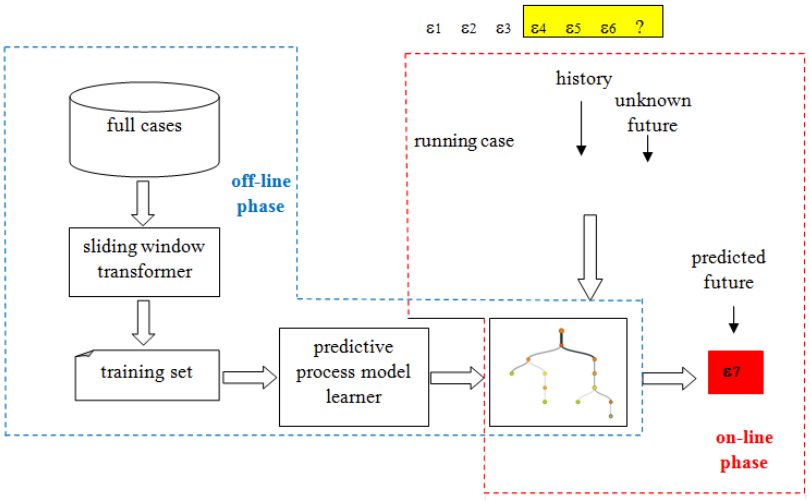


Fig. 2. Event forecasting framework. In the off-line phase, the sliding window model is used to transform the event forecasting task in a predictive clustering task: a PCT is learned from historical full traces of a process. In the on-line phase, the future event of a running case is forecast from its known past.

$$training(\mathcal{C}, w) = \{\mathbf{xy}(\mathcal{C}(i - w + 1 \rightarrow i)) | i = 1, \ldots, |\mathcal{C}|\}, \tag{4}$$

where $|\mathcal{C}|$ denotes the length of the case $\mathcal{C}$, and

$$\mathcal{T} = \bigcup_{\mathcal{C} \in \mathcal{T}} training(\mathcal{C}, w). \tag{5}$$

Each property of an event is transformed into a variable. The target space $\mathbf{Y}$ is populated with variables originated from the newest event in the sliding window, while the descriptive space $\mathbf{X}$ is populated with variables generated from the oldest $w - 1$ time-delayed events in the sliding window. The window size influences the size of the descriptive space $\mathbf{X}$. The longer the window history, the higher the number of the descriptive variables considered for the predictive clustering task. The timestamp is used when generating the descriptive space only. It is transformed into the time (in seconds) gone by the beginning of the case. When an optional property lacks in the related event, the associated variable assumes the value "none" in the training example. An example of this sliding window transformation of a case is reported in Example 1.

Example 1 (Sliding window transformation). Let us consider the case 1 of the event log reported in Table 1. The sliding window transformation of this case generates nine training examples, one for each event in the case. Each timestamp is transformed into the time (in seconds) gone by the beginning of the case.

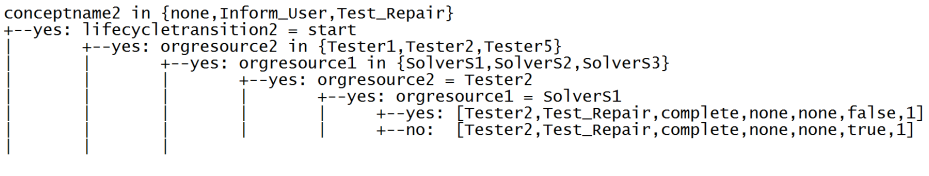


Fig. 3. A fragment of PCT learned from the event log *repair*.

By considering the window size $w = 3$, the following training examples are generated:

none, 0,none, none, none, none, none, none, none,
(1) none, 0,none, none, none, none, none, none, none,
Register, System, complete,none, none, none, none
none, 0,none, none, none, none, none, none, none,
(2) Register, 0, System, complete,none, none, none, none,
Analyze Defect, Tester3, start, none, none, none, none
Register,0,System, complete,none, none, none, none,
(3) Analyze Defect,0,Tester3,start,none, none, none, none,
Analyze Defect, Tester3, complete, T2, 6, none, none
. . .
TestRepair, 1320, Test3, complete, none, none, none, none,
(9) Inform User, 2820, System, complete, none, none, none, none,
Active Repair, System, complete, none, none, true,0

In this case, the descriptive variables are generated from the properties of the oldest two $(w-1)$ time-delayed events in the window, while the target variables (in italics) are generated from the properties of the last event of the window.

Let τ be the PCT learned from $\mathcal{T}$. It generalizes an event-based predictive process model for the process $\mathcal{P}$ (see Figure 3). In the on-line phase, this process model can be used to forecast properties of the next event in each new running case of $\mathcal{P}$. The forecast is that produced by traversing τ based on the properties of the past $w-1$ time delayed events in the case. The selected leaf contains predictions of properties for next event.

4 Empirical Study

The event-based forecasting approach described in this paper processes event logs in the XES[1] format, that is, the XML-based standard for event logs. The predictive clustering tree learner is CLUS[2] [2]. We demonstrate the applicability of the proposed approach by using three benchmark event logs[3].

[1] http://www.xes-standard.org/
[2] http://dtai.cs.kuleuven.be/clus/
[3] http://www.processmining.org/event_logs_and_models_used_in_book

4.1 Event log description

We consider event logs recorded for three different processes. These logs contain the mandatory information about activity, resource, lifecycle and time as well as process-specific optional attributes.

The event log *reviewing* handles reviews for a journal. It consists of 100 cases (papers) and 3730 events. Each paper is sent to three different reviewers. The reviewers are invited to write a report. However, reviewers often do not respond. As a result, it is not always possible to make a decision after a first round of reviewing. If there are not enough reports, then additional reviewers are invited. This process is repeated until a final decision can be made (accept or reject). The optional properties are Result by Reviewer A (accept, reject), Result by Reviewer B (accept, reject), Result by Reviewer C (accept, reject), Result by Reviewer X (accept, reject), accepts (0, 1, 2, 3, 4, 5) and rejects (0, 1, 2, 3, 4, 5). The case length ranges between 11 and 92.

The event log *repair* is about a process to repair telephones in a company. It consists of 1104 cases and 11855 events. The process starts by registering a telephone device sent by a customer. After registration, the telephone is sent to the Problem Detection Department where the defect is analyzed and classified. Once the problem is identified, the telephone is sent to the Repair Department which has two teams: one of the team fixes simple defects and the other fixes the complex defects. Once the repair employer finishes, the device is sent to the quality Assurance department. If the telephone is not repaired, it is sent again to the Repair department. Otherwise the case is archived and the telephone is sent to the customer. The company tries to fix a defect a limited number of times. The optional properties are defect type (ten categories), phone type (three categories), defect fixed (true or false) and number of repairs(0, 1, 2, 3). The case length ranges between 4 and 25.

The event log *teleclaim* contains an event log describing the handling of claims in an insurance company. It consists of 3512 cases (claims) and 46138 events. The process deals with the handling of inbound phone calls, whereby different types of insurance claims (household, car, etc.) are lodged over the phone. The process is supported by two separate call centers operating for two different organizational entities. Both centers are similar in terms of incoming call volume and average total call handling time, but different in the way call center agents are deployed. After the initial steps in the call center, the remainder of the process is handled by the back-office of the insurance company. It is noteworthy that this log is difficult to mine; the alpha algorithm fails to extract the right process model [7]. The optional properties are outcome (B insufficient information, S insufficient information, not liable, processed, rejected) and location (Brisbane and Sydney). The case length ranges between 5 and 18.

4.2 Goal and experimental set-up

The goal of this experimental study is to investigate the predictive ability of the process mining approach presented in the paper. The evaluation is performed

on the three event logs described above. The evaluation is performed in terms of several metrics, which include forecasting accuracy, model complexity, and learning time. These performance measures are estimated by using 10-fold cross validation of logs and by varying the window size between two and the maximum length of a case in the log. In particular, the accuracy is measured in terms of the classification accuracy for events of the test running cases. The model complexity is measured in terms of the number of leaves in the learned trees. The computation time is measured in seconds. The experiments were run on an Intel (R) Core(TM) i7-2670QM CPU @ 2.20 GHz server running the Windows 7 Professional.

4.3 Results and discussion

Forecasting accuracy is plotted in Figures 4(a), 4(c), 4(e), model complexity is plotted in Figures 4(b), 4(d), 4(f) 4, while learning time is plotted in Figures 5(a), 5(b), 5(c). Forecasting accuracy is averaged on the target space.

We observe that, for all logs, forecasting accuracy, as well as model complexity and learning time grow up by increasing w. While forecasting accuracy and model complexity metrics reach a (near) stable pick when $w \geq 6$, the time spent to learn a predictive process model grows-up continuously and linearly with the window size. Although the learning time is always below 7 seconds for reviewing and teleclaim cases, 4 seconds for repair cases, our study indicates that accurate forecasts can be produced by focusing the predictive analysis on the five time-delayed past events. This recommends the choice $w = 6$ which, in all logs, guarantees the highest forecasting accuracy with the lowest learning time cost. In addition, we note that, for the choice $w = 6$, the accuracy, averaged on the target space, is always high, above 90% for all logs. This valuable predictive ability of the proposed approach is confirmed by analysing the accuracy metric as it is computed for each target property individually.

Results for $w = 6$ are reported in Table 2. This finer analysis shows that our predictive model is able to forecast the majority of properties of an event with an accuracy that is greater than 92%. Low accuracy is observed only when predicting resource of events of repair cases. A deeper analysis of this process reveals that repair resource domain includes the values SolverC1, SolverC2, SolverC3, SolverS1, SolverS2, SolverS3, System, Tester1, Tester2, Tester3, Tester4, Tester5, Tester6. By grouping SolverC1, SolverC2 and SolverC3 in a SolverC category, SolverS1, SolverS2, SolverS3 in a SolverS category, as well as Tester1, Tester2, Tester3, Tester4, Tester5, Tester6 in a Tester category, the accuracy for forecasting resource passes from to 0.66 to 0.92. This means that the learned model is able, at least, to identify the resource category accurately although it is not very accurate when identifying the individual resource within the specific resource category.

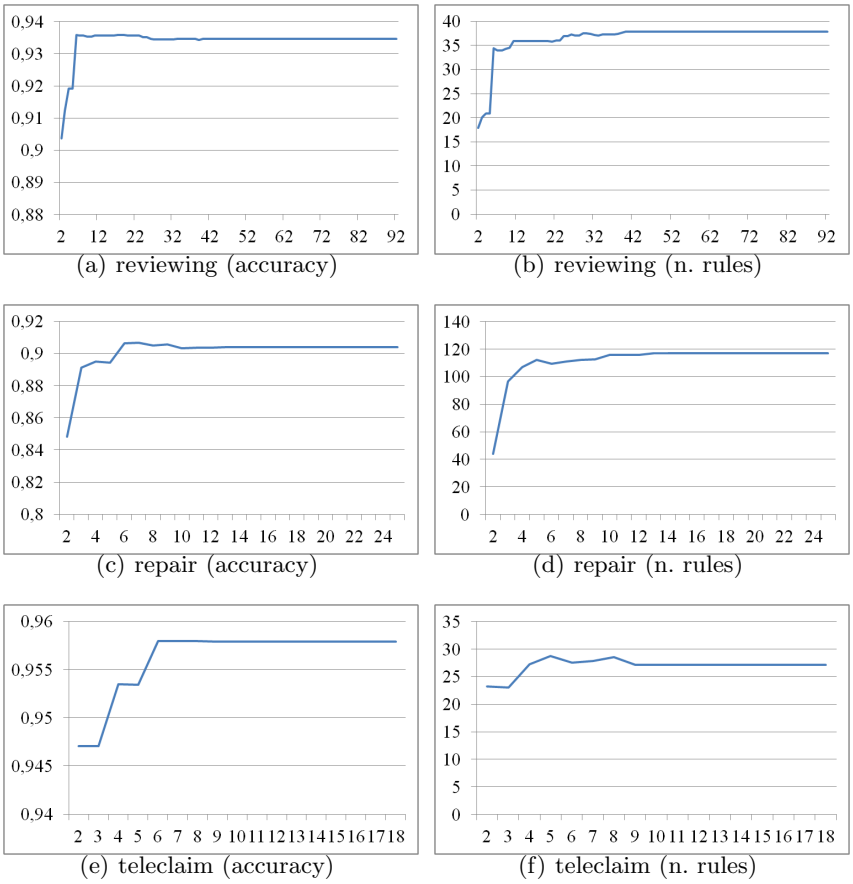
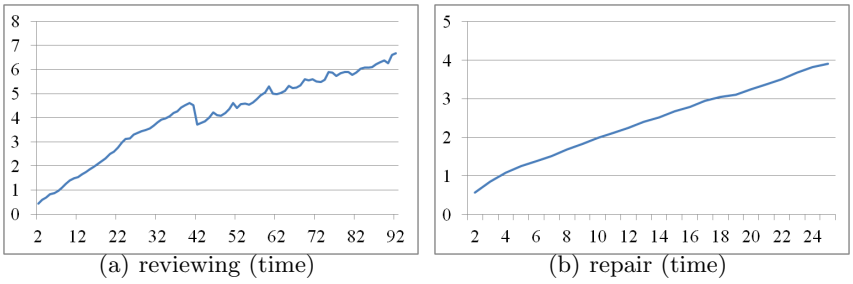


Fig. 4. Forecasting accuracy (4(a), 4(c), 4(e)) averaged on the target space and predictive model complexity (4(b), 4(d), 4(f)).

5 Conclusion

In this paper, we focus on the application of process mining to the prediction of the future of a running case. Given a running case, our prediction allows us answering questions like "what is the activity of the next event?" , "who is the resource triggering the next event?" and so on. We present a data mining framework for event-based prediction support. The framework uses a sliding window-based transformation of the event forecasting task in a predictive clustering task. A predictive process model is learned off-line and used to forecast on-line future events. Forecasts are based on the time delayed events of a running case. In the future, we would like to extend this study by using our forecasts to check conformance of running cases and recommend appropriate actions. We also plan



(a) reviewing (time) (b) repair (time)

(c) teleclaim (time)

Fig. 5. Learning time (seconds).

Table 2. Forecasting accuracy of the predictive process model learned with $w = 6$. The metric is reported for each target attribute.

repair	resource	name	lifecycle	defectType	phoneType	defectFixed	numberRepairs
	0.66	0.92	0.99	0.92	0.93	0.96	0.97

reviewing	resource	name	lifecycle	ResultA	ResultB	ResultC	ResultX	accepts	rejects
	0.79	0.82	1.00	0.99	0.99	0.99	0.93	0.96	0.96

teleclaim	resource	name	lifecycle	outcome	location
	0.96266	0.89786	1.00	0.96666	0.96266

to extend this study by using baseline methods (e.g. single-target classification methods) as well as alternative data stream models (e.g. a landmark [1] based transformation of events as well as relevant event selection [11] for training) in order to be able of accounting for older data together with the newly arriving data when learning the forecasting model.

6 Acknowledgments

This work fulfills the research objectives of the PON 02_00563_3470993 project "VINCENTE - A Virtual collective INtelligenCe ENvironment to develop sustainable Technology Entrepreneurship ecosystems" funded by the Italian Ministry of University and Research (MIUR).

References

1. C. C. Aggarwal, editor. *Data Streams - Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer, 2007.
2. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *ICML 1998*, pages 55–63. Morgan Kaufmann, 1998.
3. M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM SIGMOD Record*, 34(2):18–26, 2005.
4. R. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kauffmann, 1993.
5. F. Takens. Detecting strange attractors in turbulence. *Dynamical systems and turbulence, Warwick*, (898(1)):366–381, 1981.
6. W. van der Aalst, A. Adriansyah, A. Medeiros, F. Arcieri, T. Baier, T. Blickle, J. Bose, P. Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. Leoni, P. Delias, B. Dongen, M. Dumas, S. Dustdar, D. Fahland, D. Ferreira, W. Gaaloul, F. Geffen, S. Goel, C. Gunther, A. Guzzo, P. Harmon, A. Hofstede, J. Hoogland, J. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. Rosa, F. Maggi, D. Malerba, R. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. Motahari-Nezhad, M. Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Prez, R. Seguel Perez, M. Sepulveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. Wynn. Process mining manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer Berlin Heidelberg, 2012.
7. W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
8. W. M. P. van der Aalst, M. Pesic, and M. Song. Beyond process mining: from the past to present and future. In *the 22nd international conference on Advanced information systems engineering*, CAiSE'10, pages 38–52, Berlin, Heidelberg, 2010. Springer-Verlag.
9. W. M. P. van der Aalst, M. H. Schonenberg, and M. Song. Time prediction based on process mining. *Inf. Syst.*, 36(2):450–475, Apr. 2011.
10. H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. Xes, xesame, and prom 6. In P. Soffer and E. Proper, editors, *Information Systems Evolution - CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer, 2010.
11. I. Zliobaite. Combining time and space similarity for small size learning under concept drift. In J. Rauch, Z. W. Ras, P. Berka, and T. Elomaa, editors, *Proceedings of the 18th International Symposium on Foundations of Intelligent Systems, ISMIS 2009*, volume 5722 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 2009.

The use of the label hierarchy in HMC improves performance: A case study in predicting community structure in ecology

Jurica Levatić[1,2], Dragi Kocev[1], and Sašo Džeroski[1,2]

[1] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
Jurica.Levatic@ijs.si, Dragi.Kocev@ijs.si, Saso.Dzeroski@ijs.si

Abstract. In this article, we address the task of learning models for predicting structured outputs. We consider both global and local prediction of structured outputs, the former based on a single model that predicts the entire output structure and the latter based on a collection of models, each predicting a component of the output structure. More specifically, we investigate whether the global models have better predictive performance than the local predictive models. Moreover, we discuss the interpretability power of the obtained models. Furthermore, we evaluate the predictive models on two case studies from ecological modelling. Finally, we identify the properties of the data and the eco-system under consideration that lead to the differences in the performance.

Keywords: predictive clustering trees, hierarchical multi-label classification, multi-label classification, habitat modelling

1 Introduction

Supervised learning is one of the most widely researched and investigated areas of machine learning. The goal in supervised learning is to learn, from a set of examples with known class, a function that outputs a prediction for the class of a previously unseen example. If the examples belong to two classes (e.g., the example has some property or not) the task is called binary classification. The task where the examples can belong to a single class from a given set of m classes ($m \geq 3$) is known as multi-class classification. The case where the output is a real value is called regression.

However, in many real life problems of predictive modelling the output (i.e., the target) is structured, meaning that there can be dependencies between classes (e.g., classes are organized into a tree-shaped hierarchy or a directed acyclic graph) or some internal relations between the classes (e.g., sequences). These types of problems occur in domains such as life sciences (predicting gene function, finding the most important genes for a given disease, predicting toxicity of molecules, etc.), ecology (analysis of remotely sensed data, habitat modelling), multimedia (annotation and retrieval of images and videos) and the semantic

web (categorization and analysis of text and web pages). Having in mind the needs of these application domains and the increasing quantities of structured data, Kriegel et al. [1] and Dietterich et al. [2] listed the task of "mining complex knowledge from complex data" as one of the most challenging problems in machine learning.

A variety of methods, specialized in predicting a given type of structured output (e.g., a hierarchy of classes [3]), have been proposed [4]. These methods can be categorized into two groups of methods for solving the problem of predicting structured outputs [3, 4]: (1) local methods that predict component(s) of the output and then combine the individual models to get the overall model and (2) global methods that predict the complete structure as a whole (also known as 'big-bang' approaches). The global methods have several advantages over the local methods. First, they exploit and use the dependencies that exist between the components of the structured output in the model learning phase, which can result in better predictive performance. Next, they are typically more efficient: it can easily happen that the number of components in the output is very large (e.g., hierarchies in functional genomics can have several thousands of components), in which case executing a basic method for each component is not feasible. Furthermore, they produce models that are typically smaller than the sum of the sizes of the models built for each of the components.

Albeit the many interesting applications and the developed methods, it is not clear when it is favorable (performance wise) to construct global models and when local models. In this work, we focus on this important issue for the task of hierarchical multi-label classification (HMC). HMC is a variant of classification where a single example may belong to multiple classes at the same time and the classes are organized in a form of hierarchy. An example that belongs to some class c automatically belongs to all super-classes of c: This is called the hierarchical constraint. Problems of this kind can be found in many domains including text classification, functional genomics, and object/scene classification. Silla and Freitas [3] give a detailed overview of the possible application areas and the available approaches to HMC.

We construct four types of predictive models that exploit different amounts of the information provided by the output structure, i.e., the hierarchical organization of the classes. We investigate the predictive performance of simple single-class classification trees, hierarchical single-label classification trees, multi-label classification trees and HMC trees. The first two predictive models are local models, while the last two are global models.

The predictive models that we consider in this article are predictive clustering trees (PCTs). They can be considered as a generalization of standard decision trees towards predicting structured outputs. PCTs offer a unifying approach for dealing with different types of structured outputs and construct the predictive models very efficiently. They are able to make predictions for several types of structured outputs: tuples of continuous/discrete variables, hierarchies of classes, and time series. More details about the PCT framework can be found in [5–7].

We perform the evaluation of the predictive models on two practically relevant datasets from the task of habitat modelling [8]. Habitat modelling focuses on the spatial aspects of the distribution and abundance of plants and animals. It studies the relationships between environmental variables and the presence/abundance of plants and animals. This is typically done under the implicit assumption that both are observed at a single point in time for a given spatial unit (i.e., sampling site). We investigate the effect of environmental conditions on communities of organisms in two different ecosystems. Namely, we consider the Collembola community in the soils of Denmark [9] and organisms living in Slovenian rivers [10]. The structured output space in these case studies is the taxonomic hierarchy of the species under consideration.

The remainder of this paper is organized as follows. Section 2 explains the predictive clustering trees framework and the extensions for the different tasks considered here. The experimental setup is given in Section 3. Section 4 presents the obtained results. Finally, the conclusions are stated in Section 5.

2 Predictive modelling for HMC

In this section, we present the methodology used to construct the predictive models. We first present global predictive models that predict the complete output with a single model (i.e., a single model for all of the species present in the dataset). We then overview local predictive models that construct several models - each one predicting a part of the output (i.e., a model for each species separately).

2.1 Global predictive models

The Predictive Clustering Trees (PCTs) framework views a decision tree as a hierarchy of clusters: the top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The PCT framework is implemented in the CLUS system [11], which is available for download at `http://clus.sourceforge.net`.

PCTs are induced with a standard *top-down induction of decision trees* (TDIDT) algorithm [12]. The algorithm is presented in Table 1. It takes as input a set of examples (E) and outputs a tree. The heuristic (h) that is used for selecting the tests (t) is the reduction in variance caused by partitioning ($\mathcal{P}$) the instances (see line 4 of the BestTest procedure in Table 1). By maximizing the variance reduction, the cluster homogeneity is maximized and the predictive performance is improved.

The main difference between the algorithm for learning PCTs and a standard decision tree learner is that the former considers the variance function and the prototype function, that computes a label for each leaf, as *parameters* that can be instantiated for a given learning task. So far, PCTs have been instantiated for the following tasks: multi-target prediction (which includes multi-label classification) [6], hierarchical multi-label classification [7] and prediction of time-series [13]. In this article, we focus on the first two tasks.

Table 1. The top-down induction algorithm for PCTs.

procedure PCT
Input: A dataset E
Output: A predictive clustering tree
1: $(t^*, h^*, \mathcal{P}^*) = \text{BestTest}(E)$
2: **if** $t^* \neq none$ **then**
3: **for each** $E_i \in \mathcal{P}^*$ **do**
4: $tree_i = \text{PCT}(E_i)$
5: **return** node$(t^*, \bigcup_i \{tree_i\})$
6: **else**
7: **return** leaf(Prototype(E))

procedure BestTest
Input: A dataset E
Output: the best test (t^*), its heuristic score (h^*) and the partition $(\mathcal{P}^*)$ it induces on the dataset (E)
1: $(t^*, h^*, \mathcal{P}^*) = (none, 0, \emptyset)$
2: **for each** possible test t **do**
3: $\mathcal{P}$ = partition induced by t on E
4: $h = Var(E) - \sum_{E_i \in \mathcal{P}} \frac{|E_i|}{|E|} Var(E_i)$
5: **if** $(h > h^*) \wedge \text{Acceptable}(t, \mathcal{P})$ **then**
6: $(t^*, h^*, \mathcal{P}^*) = (t, h, \mathcal{P})$
7: **return** $(t^*, h^*, \mathcal{P}^*)$

PCTs for multi-label classification PCTs for multi-label classification can be considered as PCTs that are able to predict multiple discrete targets simultaneously. Therefore, the variance function for the PCTs for MLC is computed as the sum of the Gini indices of the target variables, i.e., $Var(E) = \sum_{i=1}^{T} Gini(E, Y_i)$. Furthermore, one can also use the sum of the entropies of class variables as a variance function, i.e., $Var(E) = \sum_{i=1}^{T} Entropy(E, Y_i)$ (this definition has also been used in the context of multi–label prediction [14]). The CLUS system also implements other variance functions, such as reduced error, gain ratio and the m-estimate. The prototype function returns a vector of probabilities that an instance belongs to a given class for each target variable. Using these probabilities, the most probable (majority) class for each target attribute can be calculated.

PCTs for hierarchical multi–label classification CLUS-HMC is the instantiation (with the distances and prototypes as defined below) of the PCT algorithm for hierarchical classification implemented in the CLUS system [7]. The variance and prototype are defined as follows. First, the set of labels of each example is represented as a vector with binary components; the i'th component of the vector is 1 if the example belongs to class c_i and 0 otherwise. It is easily checked that the arithmetic mean of a set of such vectors contains as i'th component the proportion of examples of the set belonging to class c_i. The variance of a set of examples E is defined as the average squared distance between each example's class vector (L_i) and the set's mean class vector $(\overline{L})$, i.e.,

$$Var(E) = \frac{1}{|E|} \cdot \sum_{E_i \in E} d(L_i, \overline{L})^2.$$

In the HMC context, the similarity at higher levels of the hierarchy is more important than the similarity at lower levels. This is reflected in the distance

measure used in the above formula, which is a weighted Euclidean distance:

$$d(L_1, L_2) = \sqrt{\sum_{l=1}^{|L|} w(c_l) \cdot (L_{1,l} - L_{2,l})^2},$$

where $L_{i,l}$ is the l'th component of the class vector L_i of an instance E_i, $|L|$ is the size of the class vector, and the class weights $w(c)$ decrease with the depth of the class in the hierarchy. More precisely, $w(c) = w_0 \cdot \{w(p(c))\}$, where $p(c)$ denotes the parent of class c and $0 < w_0 < 1$).

For example, consider the toy class hierarchy shown in Figure 1(a,b), and two data examples: (X_1, S_1) and (X_2, S_2) that belong to the classes $S_1 = \{c_1, c_2, c_{2.2}\}$ (boldface in Figure 1(b)) and $S_2 = \{c_2\}$, respectively. We use a vector representation with consecutive components representing membership of class c_1, c_2, $c_{2.1}$, $c_{2.2}$ and c_3, in that order (preorder traversal of the tree of class labels). The distance is then calculated as follows:

$$d(S_1, S_2) = d([1, 1, 0, 1, 0], [0, 1, 0, 0, 0]) = \sqrt{w_0 + w_0^2}.$$

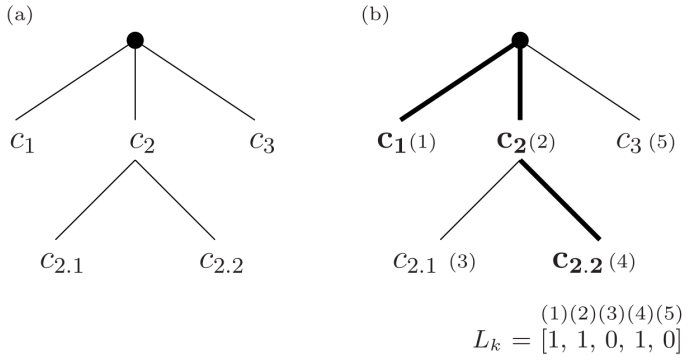


Fig. 1. Toy examples of hierarchies structured as a tree. (a) Class label names contain information about the position in the hierarchy, e.g., $c_{2.1}$ is a subclass of c_2. (b) The set of classes $S_1 = \{c_1, c_2, c_{2.2}\}$, shown in bold in the hierarchy, represented as a vector (L_k).

Recall that the instantiation of PCTs for a given task requires proper instantiation of the variance and prototype functions. The variance function for the HMC task is instantiated by using the weighted Euclidean distance measure (as given above), which is further used to select the best test for a given node by calculating the heuristic score (line 4 from the algorithm in Table 1). We now discuss the instantiation of the prototype function for the HMC task.

A classification tree stores in a leaf the majority class for that leaf, which will be the tree's prediction for all examples that will arrive in the leaf. In the

case of HMC, an example may have multiple classes, thus the notion of *majority class* does not apply in a straightforward manner. Instead, the mean $\bar{L}$ of the class vectors of the examples in the leaf is stored as a prediction. Note that the value for the i-th component of $\bar{L}$ can be interpreted as the probability that an example arriving at the given leaf belongs to class c_i.

The prediction for an example that arrives at the leaf can be obtained by applying a user defined threshold τ to the probability; if the i-th component of $\bar{L}$ is above τ then the examples belong to class c_i. When a PCT is making a prediction, it preserves the hierarchy constraint (the predictions comply with the parent-child relationships from the hierarchy) if the values for the thresholds τ are chosen as follows: $\tau_i \leq \tau_j$ whenever $c_i \leq_h c_j$ (c_i is ancestor of c_j). The threshold τ is selected depending on the context. The user may set the threshold such that the resulting classifier has high precision at the cost of lower recall or vice versa, to maximize the F-score, to maximize the interpretability or plausibility of the resulting model etc. In this work, we use a threshold-independent measure (precision-recall curves) to evaluate the performance of the HMC models.

2.2 Local habitat models

Local predictive models of structured outputs use a collection of predictive models, each predicting a component of the overall structure that needs to be predicted. The local predictive models for the task of predicting multiple targets are constructed by learning a predictive model for each of the targets separately. In the task of hierarchical multi-label classification, however, there are four different approaches that can be used: flat classification, local classifiers per level, local classifiers per node, and local classifiers per parent node (see [3] for details).

Vens et al. [7] investigated the performance of the last two approaches with local classifiers over a large collection of datasets from functional genomics. The conclusion of the study was that the last approach (called hierarchical single-label classification - HSC) performs better in terms of predictive performance, smaller total model size and faster induction times.

In particular, the CLUS-HSC algorithm by Vens et al. [7] constructs a decision tree classifier for each edge (connecting a class c with a parent class $par(c)$) in the hierarchy, thus creating an architecture of classifiers. The corresponding tree predicts membership to class c, using the instances that belong to $par(c)$. The construction of this type of trees uses few instances, as only instances labeled with $par(c)$ are used for training. The instances labeled with class c are positive instances, while the ones that are labeled with $par(c)$, but not with c are negative.

The resulting HSC tree predicts the conditional probability $P(c|par(c))$. A new instance is predicted by recursive application of the product rule $P(c) = P(c \,|\, par(c)) \cdot P(par(c))$, starting from the tree for the top-level class. Again, the probabilities are thresholded to obtain the set of predicted classes. To satisfy the hierarchy constraint, the threshold τ should be chosen as in the case of CLUS-HMC.

In this work, we also construct single-label classification trees. We construct these models by setting the number of labels to 1 and use the same algorithm as for the multi-label classification models.

3 Experimental design

In this section, we present the design of the experimental evaluation. We begin by describing the data used in the case study. Next, we outline the specific experimental setup for constructing the predictive models. Finally, we give the evaluation measure for assessing the predictive performance of the predictive models.

3.1 Data description

We use datasets from two studies that concern two eco-systems: river and soil. Namely, we construct habitat models for river water organisms living in the Slovenian rivers [10] and for soil microarthropods from Danish farms [9].

The data for the water organisms from Slovenian rivers come from the Hydrometeorological Institute of Slovenia (now Environmental Agency of Slovenia) that performs water quality monitoring for Slovenian rivers and maintains a database of water quality samples. The data provided cover a six year period of monitoring, starting from 1990 until 1995. Biological samples were taken twice a year, once in summer and once in winter, while physical and chemical samples were taken several times a year for each sampling site. In total, there are 1060 samples, each is described with 16 attributes corresponding to physical and chemical properties of water. Presence of 491 species is recorded at each sampling site, with an average of 25 species per site. Species are organized in taxonomic hierarchy with 724 nodes, with maximal depth of 4 (most general taxonomic rank is 'order').

The data for the soil microarthropods from Danish farms describes four experimental farming systems (observed during the period 1989-1993) and a number of organic farms in Denmark (observed during the period 2002-2003). Soil samples were collected within a $20m \times 20m$ area of the field, with a distance of $5m$ between the individual samples. Sampling was performed in the upper 5.5 cm soil layer and the sampling containers measured 6 cm in diameter. The data concerns the *Collembola* species community in the soil samples. These species can be used as indicators of the soil quality (in particular soil dessication) and some are considered as pests for the plants. Also, they are one of the main biological factors responsible for the control of the soil microorganisms [15]. In total, there are 1944 sites, each described with 137 attributes corresponding to various agricultural events and soil biological parameters. Presence of 35 species is recorded at each site, with an average of 7 species per site. Species are organized in taxonomic hierarchy with 72 nodes, with maximal depth of 3 (most general taxonomic rank is 'family').

3.2 Experimental design

We constructed four types of predictive models described in the previous section for each of the case studies. First, we constructed single-label classification trees for each species separately. Next, we constructed hierarchical single-label classification tree for each species. Furthermore, we constructed multi-label classification tree for all of the species, but without using the hierarchy. Finally, we constructed a hierarchical multi-label classification tree for all of the species with using the hierarchy.

We used F-test pruning to ensure that the produced models are not overfitted and have better predictive performance. This pruning procedure uses the exact Fisher test to check whether a given split/test in an internal node of the tree results in a reduction in variance that is statistically significant at a given significance level. If there is no split/test that can satisfy this, then the node is converted to a leaf. An optimal significance level was selected by using internal 3-fold cross validation, from the following values: 0.125, 0.1, 0.05, 0.01, 0.005 and 0.001.

The w_0 parameter determines the class weights with respect to the depth of the class within the hierarchy used for learning the CLUS-HMC model. We considered 3 different values of w_0: 0.75, 1 and 1.25, meaning that the classes at higher levels of hierarchy are more important, all classes are equally important and classes at lower levels of the hierarchy are more important, respectively. Different choices of w_0 yielded similar results, with $w_0 = 1$ performing slightly better. Performance measures presented in Section 4 correspond to the CLUS-HMC model learned with $w_0 = 1$.

3.3 Evaluation measures

We evaluate the algorithms using the Area Under the Precision-Recall Curve (AUPRC), and in particular, the Area Under the Average Precision-Recall Curve (AU$\overline{\text{PRC}}$) as suggested by Vens et al. [7]. The points in the PR space are obtained by varying the value for the threshold τ from 0 to 1 with step 0.02. For each value of the threshold τ, precision and recall are micro-averaged as follows:

$$\overline{Prec} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i}, \quad \text{and} \quad \overline{Rec} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i}$$

where i ranges over all classes. In the case of hierarchical classification, only the performance on the classes which correspond to leafs in the taxonomic hierarchy (i.e., species) are taken into account. To estimate the predictive performance of the obtained models, we used the 10-fold cross-validation procedure.

4 Results and discussion

In this section, we present the results from the experimental evaluation. We discuss the obtained models first by their predictive performance and then by their interpretability power.

The predictive performance of the models is given in Figure 2a. A quick inspection of the performance shows that the global models are better than the local models on the river communities study and both types of models perform equally well on the soil communities study (with the note that the HMC model performs slightly better than rest of the models). To test whether the observed differences are statistically significant, we followed the methodology proposed by Demšar [16]: Applying the Friedman test to the per-fold performance figures for each dataset separately, shows that the AU$\overline{\text{PRC}}$s are statistically different with a *p-value* $= 3.3 \times 10^{-16}$ (Slovenian rivers) and 3.5×10^{-5} (Danish farms). Figure 2b shows the average ranks for all models together, obtained with the Nemenyi post-hoc test. We can see that the HMC model performs significantly better than the Single Target models at both datasets, and better than HSC for the Slovenian rivers dataset. The Multi Target model performs significantly better than the Single Target model for the Slovenian rivers dataset. We explain the findings from a machine learning perspective and from an ecological perspective.

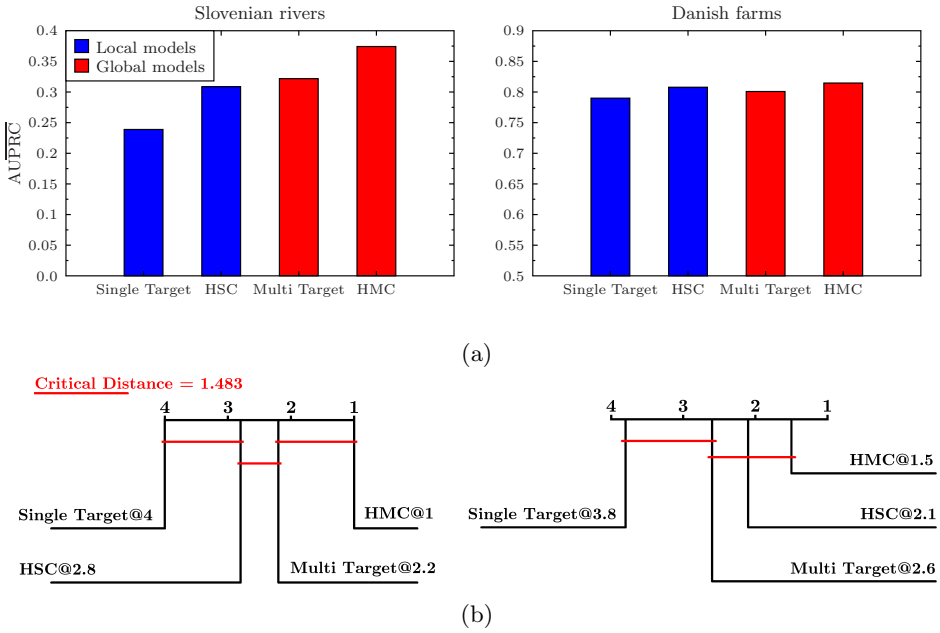


Fig. 2. (a) The area under the average precision-recall curve (AU$\overline{\text{PRC}}$) scores for the constructed predictive models. (b) Average ranks diagrams of AU$\overline{\text{PRC}}$s. Better algorithms are on the right-hand side, the ones that differ by less than critical distance for a *p-value* $= 0.05$ are connected with a horizontal bar. The number after the name of an algorithm indicates its average rank.

From a machine learning point of view, the data from the two studies have quite different properties. The average number of species per sample for the river communities (25 labels per example) is much larger than the one for the soil communities (7 labels per example). Therefore, the taxonomic hierarchy is much more populated in the former case. Arguably, such a scenario enables the HMC tree to fully exploit the hierarchy and thus produce a predictive model with a better predictive performance. We also hypothesize that the type of the descriptive attributes influenced the performance. In the study of river organisms, all of the descriptive attributes are continuous, while in the study of the soil organisms, the majority of the descriptive attributes are discrete. These continuous descriptive attributes, in this case, enable the tree construction algorithm to perform the test selection more granularly, thus obtaining more optimal tests.

From an ecological point of view, the two eco-systems studies here have different properties. More specifically, the soil eco-system is more stable than the river eco-system. This means that the soil eco-system can be more efficiently described with simpler measurements (i.e., it is much easier to monitor it). Hence, the constructed habitat models (constructed with any method) are of high quality and predictive performance (the average value of $\overline{\text{AUPRC}}$ is 0.80). Including additional information from the taxonomic ranks does not increase much the predictive performance of the constructed models. On the other hand, the river eco-system is difficult to monitor. In order to describe the state of the system, one needs to perform time-course large-scale measurements (i.e., the measurements need to cover more than a few parameters). This is the main reason because the predictive models for the river eco-system have lower predictive performance than the one from the soil eco-system (the average value of $\overline{\text{AUPRC}}$ is 0.31). Therefore, including additional information (i.e., the taxonomic ranks) helps to significantly improve the predictive performance.

In habitat modelling, besides the predictive power of the models, their interpretability is also a highly desired property. The predictive models that we consider here (PCTs) are readily interpretable. However, the difference in the interpretability of the local and global models is easy to notice. In Figure 3, we present illustrative examples of the predictive models for the Slovenian rivers dataset. We show the PCTs for single-label classification, multi-label classification and hierarchical multi-label classification.

We can immediately notice the different between the local and global predictive models. The local models[3] offer an information only for a part for the output space, i.e., they are valid just for a single species. In order to reconstruct the complete community model, one needs to look at the separate models and then try to make some overall conclusions. However, this could be very tedious or even impossible in domains with high biodiversity and where there are hundreds of species present, such as the domain we consider here - Slovenian rivers.

[3] Note that the hierarchical single-label classification models will be much similar to the single-label classification models, with the difference that the predictive models are organized into an hierarchical architecture. This makes the interpretation of the HSC models even more difficult task.

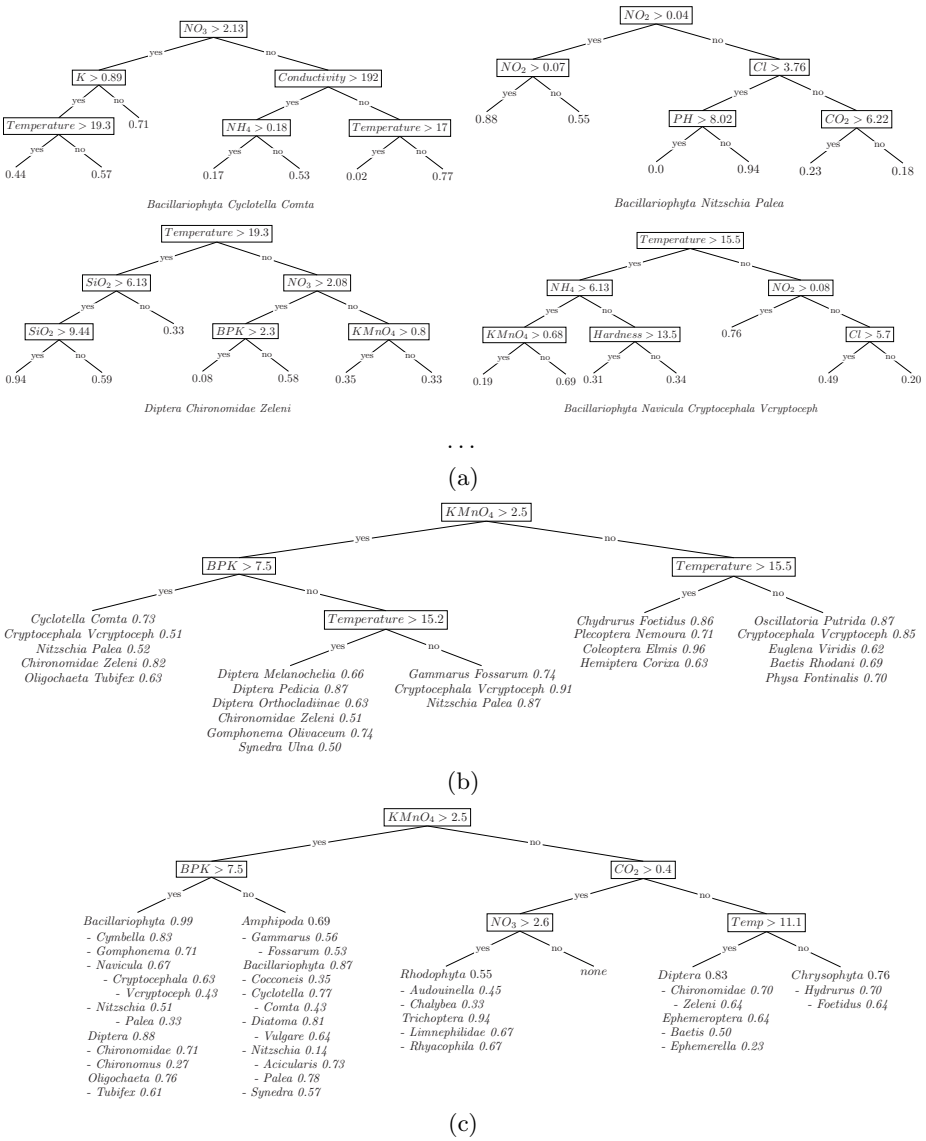


Fig. 3. Illustrative examples of decision trees for Slovenian rivers dataset constructed with PCTs. Single target classification (a) produces a separate model for each of the species, whereas multi target classification (b) and hierarchical multi-label classification (c) consider all of the species in single tree.

On the other hand, the global models are much easier to interpret. The single global model is valid for the complete structured output, i.e., for the whole community of species present in the ecosystem. The global models are able to capture the interactions present between the species, i.e., which species can co-exist at

a locations with given physico-chemical properties. Moreover, the HMC models, as compared to the multi-label models, offer additional information about the higher taxonomic ranks. For example, the HMC model could state that there is a low chance that the species *Diptera chironomus* could be present under the given environmental conditions, however the genus *Diptera* could be.

5 Conclusions

In this article, we address the task of learning predictive models for structured output learning, which takes as input a tuple of attribute values and produces a structured object. In contrast to standard classification and regression, where the output is a single scalar value, in structured output learning the output is a data structure, such as a tuple or a directed acyclic graph. We consider both global and local prediction of structured outputs, the former based on a single model that predicts the entire output structure and the latter based on a collection of models, each predicting a component of the output structure.

We investigate the differences in performance and interpretability of the local and global models. More specifically, we research whether including information in the form of a taxonomic rank helps to improve the predictive performance of the predictive models. We compare the performance of local and global predictive models on a practically relevant task from ecology - habitat modelling.

The results show that the global models perform better than the local models. This performance improvement is more pronounced on domains that have more populated hierarchy. On the other hand, the improvement is less visible on well described and stable domains where any predictive model has good predictive performance. Furthermore, the global models are much easier to interpret than the local models and offer an overview of the complete eco-system.

We plan to extend this work along several dimensions. We will start by including more datasets with different properties in the evaluation procedure. Next, we will generate artificial datasets to further check the results of this study. Finally, we will include other types of local and global models to check whether these findings carry over other predictive modelling methods.

References

1. Kriegel, H.P., Borgwardt, K., Kröger, P., Pryakhin, A., Schubert, M., Zimek, A.: Future trends in data mining. Data Mining and Knowledge Discovery **15** (2007) 87–97
2. Dietterich, T.G., Domingos, P., Getoor, L., Muggleton, S., Tadepalli, P.: Structured machine learning: the next ten years. Machine Learning **73**(1) (2008) 3–23
3. Silla, C., Freitas, A.: A survey of hierarchical classification across different application domains. Data Mining and Knowledge Discovery **22**(1-2) (2011) 31–72
4. Bakır, G.H., Hofmann, T., Schölkopf, B., Smola, A.J., Taskar, B., Vishwanathan, S.V.N.: Predicting structured data. Neural Information Processing. The MIT Press (2007)

5. Blockeel, H.: Top-down induction of first order logical decision trees. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium (1998)
6. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Tree ensembles for predicting structured outputs. Pattern Recognition **46**(3) (2013) 817–833
7. Vens, C., Struyf, J., Schietgat, L., Džeroski, S., Blockeel, H.: Decision trees for hierarchical multi-label classification. Machine Learning **73**(2) (2008) 185–214
8. Džeroski, S.: Machine learning applications in habitat suitability modeling. In Haupt, S.E., Pasini, A., Marzban, C., eds.: Artificial Intelligence Methods in the Environmental Sciences. Springer (2009) 397–412
9. Demšar, D., Džeroski, S., Larsen, T., Struyf, J., Axelsen, J., Bruns-Pedersen, M., Krogh, P.H.: Using multi-objective classification to model communities of soil. Ecological Modelling **191**(1) (2006) 131–143
10. Džeroski, S., Demšar, D., Grbović, J.: Predicting chemical parameters of river water quality from bioindicator data. Applied Intelligence **13**(1) (2000) 7–17
11. Blockeel, H., Struyf, J.: Efficient algorithms for decision tree cross-validation. Journal of Machine Learning Research **3** (2002) 621–650
12. Breiman, L., Friedman, J., Olshen, R., Stone, C.J.: Classification and Regression Trees. Chapman & Hall/CRC (1984)
13. Slavkov, I., Gjorgjioski, V., Struyf, J., Džeroski, S.: Finding explained groups of time-course gene expression profiles with predictive clustering trees. Molecular BioSystems **6**(4) (2010) 729–740
14. Clare, A.: Machine learning and data mining for yeast functional genomics. PhD thesis, University of Wales Aberystwyth, Aberystwyth, Wales, UK (2003)
15. Ponge, J.F.: Food resources and diets of soil animals in a small area of Scots pine litter. Geoderma **49**(1-2) (1991) 33–62
16. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research **7** (2006) 130

Mining Audio Data for Multiple Instrument Recognition in Classical Music

Elżbieta Kubera[1] and Alicja A. Wieczorkowska[2]

[1] University of Life Sciences in Lublin, Akademicka 13, 20-950 Lublin, Poland
elzbieta.kubera@up.lublin.pl
[2] Polish-Japanese Institute of Information Technology,
Koszykowa 86, 02-008 Warsaw, Poland
alicja@poljap.edu.pl

Abstract. This paper addresses the problem of identification of multiple musical instruments in polyphonic recordings of classical music. A set of binary random forests was used as a classifier, and each random forest was trained to recognize the target class of sounds. Training data were prepared in two versions, one based on single sounds and their mixes, and the other based on sound frames taken from classical music recordings. The experiments on identification of multiple instrument sounds in recordings are presented, and their results are discussed in this paper.

Key words: Music Information Retrieval, Sound Recognition, Random Forests

1 Introduction

Music information retrieval (MIR) became a topic of broad interest for researchers several years ago, see e.g. [14], [17], and one of the most challenging tasks within this area is to automatically extract meta-information from audio waveform [10]. Audio data stored as sound amplitude values changing over time represent very complex data, where multiple sounds of a number of instruments are represented by a single value (i.e. amplitude value of a complex sound) in each time instant in the case of monophonic recordings, or by a single value in each recording channel. Extraction of information about timbre of particular sounds is difficult, but it has been addressed in audio research last years. Identification of music titles through query-by-example, including excerpts replayed on mobile devices, has been quite successfully addressed [16], [19], as well as finding pieces of music through query-by-humming [11]. However, identification of instruments in audio excerpts is still a challenge, sometimes addressed through multi-pitch tracking [5], often supported with external provision of pitch data, or limited to identification of predominant sounds [2].

In this paper, we deal with identification of multiple sounds of multiple instruments in the recordings of classical music. No pitch tracking is required, and the classification is performed on data as is. There are no pre-assumptions regarding number of instruments in polyphony, and the recordings can contain

any instrument sounds, including instruments for which our classifiers are not trained. This is because we use a set of binary classifiers, where each classifier is trained to identify a target sound class. If none of the classifiers recognizes its target class, it means that an unknown instrument or instruments play in the analyzed audio sample. Classification is performed for mono or stereo input data, and a mix (average) of the channels is taken as input in the case of stereo recordings.

1.1 Audio Data Classification

Automatic identification of musical instruments has been performed so far by many researchers, and usually on a different set of instruments, number of classes, sound parametrization, number of sounds used, and classifiers applied. Identification of a single instrument in a single sound is the easiest case, and virtually all available classification tools have been applied for this purpose, including k-nearest neighbors, neural networks, support vector machines, rough set based classifiers, decision trees and random forests. Quality of the recognition depends heavily on the number of sounds and instruments/classes applied, and it can even reach 100% for a few classes, or be as low as 40% if there are 30 or more classes; for detailed review see [6].

Identification of instruments in polyphonic environment is much more challenging, and it has been addressed in various ways. Usually initial assumptions are made: on the number of instruments in the polyphony, on pitch data as input, on the instrument set in the analyzed recordings, or on identifying predominant sound, see e.g. [2], [5]. Since the final goal of such research is score extraction, such assumptions are understandable, and in some cases the research addresses sound separation into single sounds. However, these external data are often manually provided, not extracted from audio recordings. In our research, we would like to perform instrument recognition without any pre-assumptions, and also without initial data segmentation. We have already performed similar research, for jazz recordings [8], but it required tedious segmentation and labeling of small frames of the recordings in order to obtain ground-truth data. In order to facilitate research, we decided to perform annotation for 0.5-second excerpts; MIDI files and scores were used as guidance. Classification was performed using a set of random forests, since such a classifier proved quite successful in our previous research [8].

2 Random Forests

A random forest (RF) is a classifier based on a tree ensemble; such classifiers are gaining increasing popularity last years [15]. RF is constructed using procedure minimizing bias and correlations between individual trees. Each tree is built using a different N-element bootstrap sample of the N-element training set. The elements of the sample are drawn with replacement from the original set. Therefore, roughly 1/3 of the training data are not used in the bootstrap

sample for any given tree. Assuming that objects are described by a vector of K attributes (features), k attributes out of all K attributes are randomly selected ($k \ll K$, often $k = \sqrt{K}$) at each stage of tree building, i.e. for each node of any particular tree in RF. The best split on these k attributes is used to split the data in the node.

The best split of the data in the node is determined as minimizing the Gini impurity criterion, which is the measure of how often an element would be incorrectly labeled if labeled randomly, according to the distribution of labels in the subset. Each tree is grown to the largest extent possible, without pruning. By repeating this randomized procedure M times a collection of M trees is obtained, constituting a random forest. Classification of each object is made by simple voting of all trees [1].

The classifier used in our research consists of a set of binary random forests. Each RF is trained to identify the target sound class, representing an instrument (or silence), whether this particular timbre is present in the sound frame under investigation, or not. If the percentage of votes of the trees in the RS is 50% or more, then the answer of the classifier is considered to be positive, otherwise it is considered to be negative.

3 Audio Data

Our experiments focused on musical instrument sounds of definite pitch, but information about pitch was not used not retrieved in our research. Sounds of definite pitch are produced by chordophones (stringed instruments) and aerophones (wind instruments). In our experiments, we chose wind and stringed instruments, played in various ways, i.e. with various articulation, including bowing vibrato and pizzicato. Percussive instruments, i.e. idiophones and membranophones (basically drums) were excluded from the described research. Additionally, a class representing silence was added to the set of classes representing instruments. Therefore, if none of the classifiers gives positive answer, then we can conclude that an unknown instrument or instruments are playing in the investigated sound frame.

The following sound classes were investigated in the reported research:

- flute (*fl*),
- oboe (*ob*),
- bassoon (*bn*),
- clarinet (*cl*),
- French horn (*fh*),
- violin (*vn*),
- viola (*va*),
- cello (*cl*),
- double bass (*db*),
- piano (*pn*), and
- silence (*sc*).

All sounds were recording at 44.1 kHz sampling rate with 16-bit resolution, or converted to this format.

3.1 Training Data

Training of the classifiers was performed in two versions. The first training ($T1$) was based on single sounds of musical instruments, taken from RWC [4] sets of single sounds of musical instruments, and also mixes of up to three instrument sounds were added to this training set. Single sounds of musical instruments from MUMS [13] and IOWA [18] repositories were used for mixing. Ten thousands of 40-ms long audio frames represented positive examples for each RF (i.e. frames where the target instrument is playing), with 5,000 representing single sounds and 5,000 representing mixes, and ten thousands of 40-ms long audio frames represented negative examples (i.e. frames where the target instrument is not playing). Mixes constitute a single chord or unison, and a set of instruments is always typical for classical music.

The second training ($T2$) was based on sound taken from recordings, with no initial segmentation to separate single sounds. The recordings were taken from RWC Classical Music Database [3], recordings of Mozart concerto for Flute in G-major (KV 313), for Oboe in C-major (KV 314), for Bassoon in B-flat Major (KV 191), and the following .mp3 files (converted to .au format): viola Suite No. 1 in G-major BWV 1007, J.S. Bach, for cello solo transcribed for viola (Prelude and Allemande, [9]), Suite no 1 in G-major BWV 1007, J.S. Bach, for cello transcribed for double bass, Clarinet Concerto in A-major KV622, W.A. Mozart - Allegro and Adagio, Horn Concerto in E-flat Major KV 495, W.A. Mozart - Allegro moderato. Solo and polyphonic segments were taken as training data. This training set represents more realistic sounds, which can be encountered in all classical music recordings, including their compressed file version.

3.2 Testing Data

Testing was performed on RWC Classical Music Database recordings [3], and for presentation purposes the first minute of each investigated piece was used. The following pieces were used:

- No. 1, F.J. Haydn, Symphony no.94 in G major, Hob.I-94 'The Surprise'. 1st mvmt., with the following instruments playing in the first minute of the recording: flute, oboe, bassoon, French horn, violin, viola, cello, double bass;
- No. 2, W.A. Mozart, Symphony no.40 in G minor, K.550. 1st mvmt. with the following instruments playing in the first minute of the recording: flute, oboe, bassoon, French horn, violin, viola, cello, double bass;
- No. 16, W.A. Mozart, Clarinet Quintet in A major, K.581. 1st mvmt., with the following instruments playing in the first minute of the recording: clarinet, violin, viola, cello;
- No. 18, J. Brahms, Horn Trio in Eb major, op.40. 2nd mvmt., with the following instruments playing in the first minute of the recording: piano, French horn, violin;
- No. 44, N. Rimski-Korsakov, The Flight of the Bumble Bee, with flute and piano.

These pieces represent various polyphony and pose diverse difficulties for the classifier, including short sounds, and multiple instruments playing at the same time. No training data were used in our tests.

4 Feature Set

Audio data are usually parameterized before classification is applied, since raw data representing amplitude changes vs. time undergo dramatic changes in a fraction of second, and the amount of the data is overwhelming. The identification of musical instruments in audio data depends on the sound parametrization applied, and there is no one feature set used worldwide; each research group utilizes a different feature set. Still, some features are commonly used, and our feature set is also based on these features. We decided to utilize a feature set which proved successful in our previous, similar research [8]. Our parametrization is performed for 40-ms frames of audio data. No data segmentation or pitch extraction are needed, thus multi-pitch extraction is avoided, and no labeling particular sounds in polyphonic recording with the appropriate pitches is needed. The feature vector consists of basic features, describing properties of an audio frame of 40 ms, and additionally difference features, calculated as the difference between the feature calculated for a 30 ms sub-frame starting from the beginning of the frame, and a 30 ms sub-frame starting with 10 ms offset. Fourier transform was used to calculate spectral features, with Hamming window. Most of the features we applied represent MPEG-7 low-level audio descriptors, often used in audio research [7]. Identification of instruments is performed frame by frame, for consequent frames, with 10 ms hop size. Final classification result is calculated as an average of classifier output over 0.5-second segment of the recording, in order to avoid tedious labeling of ground-truth data over shorter frames.

The feature vector we applied consists of the following 91 parameters [8]:

- *Audio Spectrum Flatness*, $flat_1, \ldots, flat_{25}$ — a multidimensional parameter describing the flatness property of the power spectrum within a frequency bin for selected bins; 25 out of 32 frequency bands were used;
- *Audio Spectrum Centroid* — the power weighted average of the frequency bins in the power spectrum; coefficients are scaled to an octave scale anchored at 1 kHz [7];
- *Audio Spectrum Spread* — RMS (root mean square) value of the deviation of the log frequency power spectrum wrt. *Audio Spectrum Centroid* [7];
- *Energy* — energy (in log scale) of the spectrum of the parametrized sound;
- *MFCC* — a vector of 13 mel frequency cepstral coefficients. The cepstrum was calculated as the logarithm of the magnitude of the spectral coefficients, and then transformed to the mel scale, to better reflect properties of the human perception of frequency. 24 mel filters were applied, and the obtained results were transformed to 12 coefficients. The 13^{th} coefficient is the 0-order coefficient of MFCC, corresponding to the logarithm of the energy [12];

– *Zero Crossing Rate*; a zero-crossing is a point where the sign of the time-domain representation of the sound wave changes;
– *Roll Off* — the frequency below which an experimentally chosen percentage equal to 85% of the accumulated magnitudes of the spectrum is concentrated; parameter originating from speech recognition, where it is applied to distinguish between voiced and unvoiced speech;
– *NonMPEG7 - Audio Spectrum Centroid* — a linear scale version of *Audio Spectrum Centroid*;
– *NonMPEG7 - Audio Spectrum Spread* — a linear scale version of *Audio Spectrum Spread*;
– changes (measured as differences) of the above features for a 30 ms sub-frame of the given 40 ms frame (starting from the beginning of this frame) and the next 30 ms sub-frame (starting with 10 ms shift), calculated for all the features shown above;
– *Flux* — the sum of squared differences between the magnitudes of the DFT points calculated for the starting and ending 30 ms sub-frames within the main 40 ms frame; this feature by definition describes changes of magnitude spectrum, thus it is not calculated in a static version.

Audio data were in mono or stereo format; mixes of the left and right channel (i.e. the average value of samples in both channels) were taken if the audio signal was in stereo format.

5 Experiments and Results

The experiments aimed at investigating how many instruments can be identified correctly in real polyphonic recordings, and whether adding real recordings representing solos and polyphonic recordings (rather than isolated single sounds and their mixes) can improve the performance of the classifier. The main problem with such classification is the recall, which is usually quite low, i.e. instruments in recordings are missed by classifiers. Another problem is how to assess the results, since many instruments can be playing in the same segment. Since we deal with binary classifiers, possible errors include false negatives (missed target instrument) and false positives (false indication of the target instrument, not playing in a given segment). The details of classification results for both training versions, $T1$ and $T2$ are shown in Table 1 and Table 2. Precision, recall, f-measure and accuracy were calculated as follows, on the basis of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN):

– precision pr was calculated as $pr = TP/(TP + FP)$,
– recall rec was calculated as $rec = TP/(TP + FN)$,
– f-measure f_{meas} was calculated as $f_{meas} = 2 \cdot pr \cdot rec/(pr + rec)$,
– accuracy acc was calculated as $acc = (TP + TN)/(TP + TN + FP + FN)$.

If the denominator in the formula for calculating precision is equal to zero, then the classifier made no error, and the precision is equal to one. Also, if the

denominator in the formula for calculating recall is equal to zero, then the recall is equal to one.

As we can see, the classifier built for the training on single sounds and mixes ($T1$) gives few indications of the target classes; therefore, the precision is often equal to one (100%). The recall is quite low, but we are glad to observe that using real unsegmented recordings for training ($T2$) improves the recall significantly.

Table 1. Results of the recognition of musical instruments in RWC Classical Music Database, for training on single sounds and their mixes ($T1$)

Result	*bn*	*ob*	*cl*	*fl*	*fh*	*pn*	*cl*	*va*	*vn*	*db*	*sc*	Average
TP	2	1	6	21	0	23	18	14	0	22	17	
FP	52	0	0	0	5	3	5	11	0	13	40	
FN	114	98	41	153	168	207	192	227	394	115	2	
TN	432	501	553	426	427	367	385	348	206	450	541	
precision	4%	100%	100%	100%	0%	88%	78%	56%	100%	63%	30%	65%
recall	2%	1%	13%	12%	0%	10%	9%	6%	0%	16%	89%	14%
f-measure	2%	2%	23%	22%	0%	18%	15%	11%	0%	26%	45%	15%
accuracy	72%	84%	93%	75%	71%	65%	67%	60%	34%	79%	93%	72%

Table 2. Results of the recognition of musical instruments in RWC Classical Music Database, for training on sounds from real recordings, representing solos and polyphonic segments ($T2$)

Result	*bn*	*ob*	*cl*	*fl*	*fh*	*pn*	*cl*	*va*	*vn*	*db*	*sc*	Average
TP	54	7	4	150	23	50	142	99	327	65	17	
FP	90	9	218	87	85	25	58	85	3	45	32	
FN	62	92	43	24	145	180	68	142	67	72	2	
TN	394	492	335	339	347	345	332	274	203	418	549	
precision	38%	44%	2%	63%	21%	67%	71%	54%	99%	59%	35%	50%
recall	47%	7%	9%	86%	14%	22%	68%	41%	83%	47%	89%	47%
f-measure	42%	12%	3%	73%	17%	33%	69%	47%	90%	53%	50%	44%
accuracy	75%	83%	57%	82%	62%	66%	79%	62%	88%	81%	94%	75%

Illustration of the results for the first piece of music (No. 1) is shown in Figure 1 for the training $T1$, and for comparison the details of recognition results for the same piece for the training $T2$ is shown in Figure 2. As we can see, if the classifier trained on $T1$ (single sounds and mixes) shows positive outcome, the indication is just above the 0.5 threshold. The outcomes for the classifier trained on $T2$ (real solo and polyphonic recordings) are much higher, although we can see errors, especially for string instruments. However, this piece is difficult for recognition as an orchestral piece of high polyphony, so errors in this case were rather unavoidable.

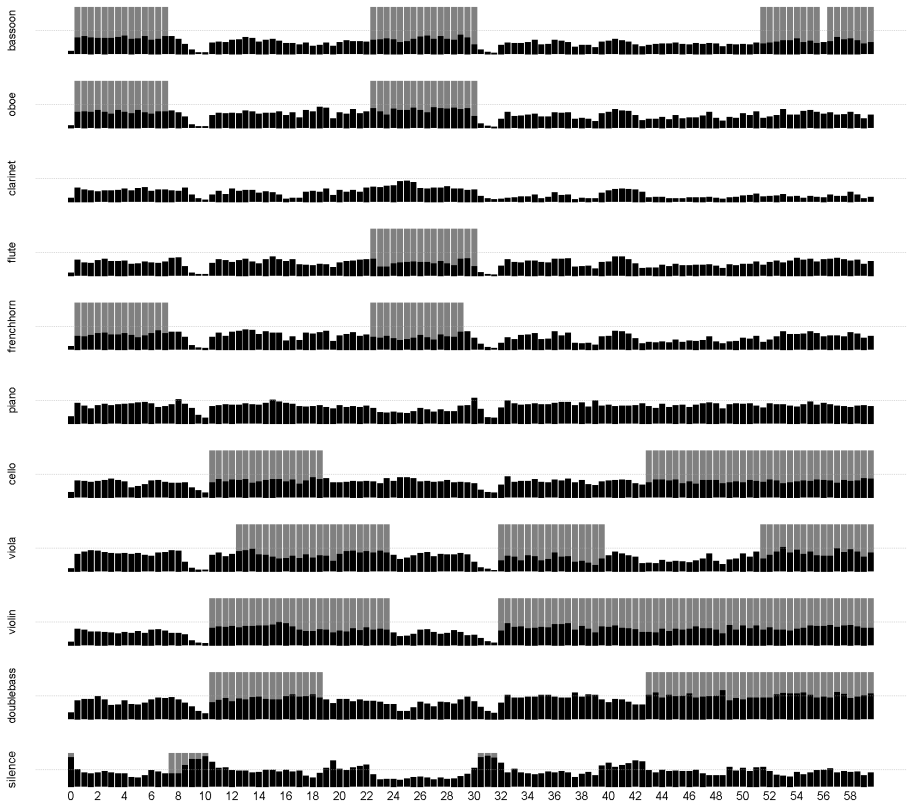


Fig. 1. Outcome of each random forest for the RWC Classical Music No. 1, for each 0.5-second segment of the first minute of the recording, for the training $T1$. If the result for a forest (trained to recognize a target instrument, or silence) is 0.5 or more, then this classifier indicates that the target instrument is playing in this segment. Ground-truth data are marked in grey.

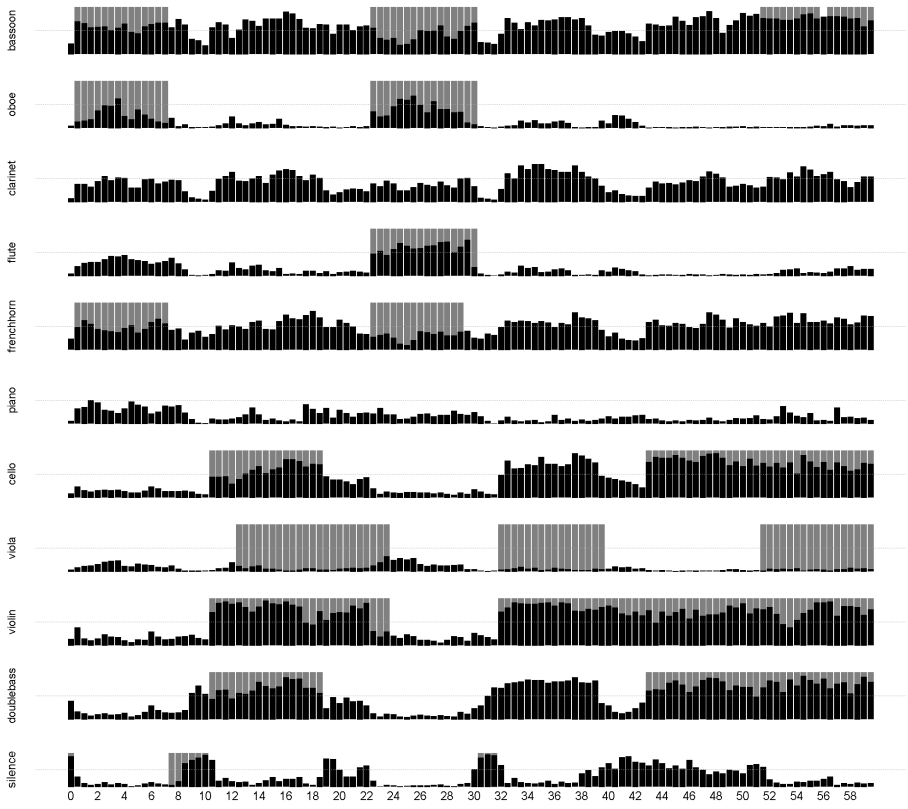


Fig. 2. Outcome of each random forest for the RWC Classical Music No. 1, for each 0.5-second segment of the first minute of the recording, for the training $T2$. If the result for a forest (trained to recognize a target instrument, or silence) is 0.5 or more, then this classifier indicates that the target instrument is playing in this segment. Ground-truth data are marked in grey.

6 Summary and Conclusions

The research presented in this paper aimed at the difficult task of recognizing multiple instruments in polyphonic recordings of classical music. Ten instrumental classes were investigated, and also silence was added as a separate class. The training was performed for single instrumental sounds and their mixes, and the second classifier was trained for excerpts taken from recordings, without segmentation. A set of binary random forests was used as a classifier, where each forest was trained to recognize whether the target instrument (or silence) is recorded in the analyzed excerpt. Forty-millisecond segments were analyzed, but the results were presented for 0.5-second segment, in order to avoid tedious labeling of ground-truth data. The results show that training performed on unsegmented real recordings improves the recall dramatically. Therefore we conclude that the training data should be adjusted to the target in hand, thus allowing the classifier to learn the sounds in typical recording set-up.

Acknowledgments. This project was partially supported by the Research Center of PJIIT, supported by the Polish Ministry of Science and Higher Education.

References

1. Breiman, L.: Random Forests. Machine Learning 45, 5–32 (2001)
2. Fuhrmann, F.: Automatic musical instrument recognition from polyphonic music audio signals. PhD Thesis, Universitat Pompeu Fabra (2012)
3. Goto, M., Hashiguchi, H., Nishimura, T., Oka, R.: RWC Music Database: Popular, Classical, and Jazz Music Databases. In: Proceedings of the 3rd International Conference on Music Information Retrieval, 287–288 (2002)
4. Goto, M., Hashiguchi, H., Nishimura, T., Oka, R.: RWC Music Database: Music Genre Database and Musical Instrument Sound Database. In: Proceedings of the 4th International Conference on Music Information Retrieval ISMIR, 229–230 (2003)
5. Heittola, T., Klapuri, A., Virtanen, A.: Musical Instrument Recognition in Polyphonic Audio Using Source-Filter Model for Sound Separation. In: Proc. 10th Int. Society for Music Information Retrieval Conf. (ISMIR 2009) (2009)
6. Herrera-Boyer, P., Klapuri, A., Davy, M.: Automatic Classification of Pitched Musical Instrument Sounds. In: Klapuri, A., Davy, M. (eds.) Signal Processing Methods for Music Transcription. Springer Science+Business Media LLC (2006)
7. ISO: MPEG-7 Overview, `http://www.chiariglione.org/mpeg/`
8. Kubera, E., Kursa, M.B., Rudnicki, W.R., Rudnicki, R., Wieczorkowska, A.A.: All That Jazz in the Random Forest. In: Kryszkiewicz, M., Rybiński, H., Skowron, A., Raś, Z.W. (eds.): ISMIS 2011. LNAI, vol. 6804, pp. 543–553. Springer, Heidelberg (2011)
9. Kuperman, M.: Suite N 1 in G-Dur BWV 1007, `http://www.viola-bach.info/`
10. Martin, K.D.: Toward automatic sound source recognition: Identifying musical instruments. Presented at the 1998 NATO Advanced Study Institute on Computational Hearing, Il Ciocco, Italy (1998)
11. MIDOMI: Search for Music Using Your Voice by Singing or Humming, `http://www.midomi.com/`

12. Niewiadomy, D., Pelikant, A.: Implementation of MFCC vector generation in classification context. J. Applied Computer Science, Vol. 16, No. 2, pp. 55–65 (2008)
13. Opolko, F., Wapnick, J.: MUMS — McGill University Master Samples. CD's (1987)
14. Ras, Z.W., Wieczorkowska, A.A. (eds.): Advances in Music Information Retrieval. Series: Studies in Computational Intelligence, Vol. 274, Springer (2010)
15. Richards, G., Wang, W.: What influences the accuracy of decision tree ensembles? J. Intell. Inf. Syst. 39, 627-650 (2012)
16. Shazam Entertainment Ltd, http://www.shazam.com/
17. Shen, J., Shepherd, J., Cui, B., Liu, L. (eds.): Intelligent Music Information Systems: Tools and Methodologies. Information Science Reference, Hershey (2008)
18. The University of IOWA Electronic Music Studios: Musical Instrument Samples, http://theremin.music.uiowa.edu/MIS.html
19. TrackID — Sony Smartphones, http://www.sonymobile.com/global-en/support/faq/xperia-x8/internet-connections-applications/trackid-ps104/

A Hybrid Distance-based Method and Support Vector Machines for Emotional Speech Detection

Vladimer Kobayashi

Université Jean Monnet
Laboratoire Hubert Curien CNRS, UMR 5516
18 rue du Pr. Benoit Lauras, 42000 Saint-Etienne, France
vladimer.kobayashi@univ-st-etienne.fr

Abstract. We describe a novel methodology that is applicable in the detection of emotions from speech signals. The methodology is useful if we can safely ignore sequence information since it constructs static feature vectors to represent a sequence of values; this is the case of the current application. In the initial feature extraction part, the speech signals are cut into speech segments of specified duration. The speech segments are processed and described using features such as pitch, energy, mel frequency cepstrum coeffiecients and linear prediction cepstrum coefficients. Our proposed methodology consists of two steps. The first step constructs emotion models using principal component analysis and it computes distances of the observations to each emotion models. The distance values from the previous step are used to train a support vector machine classifier that can identify the affective content of a speech signal. We note that our method is not only applicable for speech signal, it can also be used to analyse other data of similar nature. The proposed method is tested using two emotional databases. Results showed competitive performance yielding an average accuracy of greater than 90% on both databases for the detection of three emotions.

Key words: emotion recognition from speech, support vector machines, speech segment-level analysis

1 Introduction

The advent of Ubiquitous Computing research has paved the way to the development of systems that are more accustomed and able to respond in a timely manner according to human needs and behaviour [1, 2]. Computers and other machines have been successfully integrated to every aspect of life. To be truly practical machines must not only "think" but also "feel" since meaningful experiences are communicated through changes in affective states or emotions. At the heart of all of this is the type of data that we will handle to proceed with the computational task. In a typical scenario we deal with data types commonly encountered in signal processing since emotions are either overtly expressed in voice signals or covertly in biological signals and these signals can be captured

through the use of sophisticated sensors. The challenge not only lies on the pre-processing part but also on the development of methods adapted to the nature of data we wish to analyse.

In the past decade we have seen the explosion of studies that try to deduce human emotions from various signals we collect. By far the most carefully studied signal for this purpose is the speech signal [3, 4]. It is an accepted fact that it is relatively easy for us humans to identify (to a certain degree) the emotion of another person based on his voice, although, we are still trying to understand how we manage to do it. There are features in speech signals which we unknowingly distinguish and process that enable us to detect certain types of emotions. The central idea of many researches is to automate the process of detecting emotions which is vital to the creation of emotion-aware technologies and systems [4].

In this paper, as a first step, we also deal with speech signals. We argue that speech signals are the most convenient and reasonable to deal with since in real setting they can be easily captured. Unlike other signals such as ECG and EEG, speech signals are not particularly troublesome to collect and can be recorded anywhere and at any time. Also, many studies were published about processing of speech signals thus we can try the features proposed in previous studies in this work. The true nature of a speech signal is dynamic, a sequence of values indexed by time, however, the approach that we follow is to represent it as a single static feature vector. This approach is influenced by the fact that the sequence information is not essential for emotion recognition.

The objective of this work is the proposal of a novel technique that predicts emotions from speech signal. In essence our proposed method consisted of two steps. The first step is the creation of *emotion models* and the computation of deviations or distances of the speech signal "parts" from each of the emotion models. The second stage is to use the distance values computed in the previous step to construct a classifier that can detect the over-all emotional content of a given speech signal. In contrast with other techniques, we obtained additional knowledge such as the importance of different variables and the degree of separation of the speech signal components from each emotion categories. The first step is reminiscent of the method called Soft Modelling of Class Analogies (SIMCA) [5] although we do not attempt to classify the speech signals in this step. Another advantage of our technique is the tremendous flexibility it offers. Among other things, the modeller has the freedom to use different sets of features in both the first and second steps and adjust the underlying methods to make it robust to noise.

As a primary application of our proposed approach we deal with the problem of detecting three types of emotions, namely, "Disgust", "Angry", and "Sadness". These emotions are commonly encountered in application such as the analysis of telephone conversation and diagnosis of certain medical disorders. We tested our approach on two emotional speech databases. The results showed the effectiveness of our approach based from the analysis using the two databases. The accuracies are at least 90% on both databases.

The rest of the paper is organized as follows. Section 2 provides description of the two speech databases to which we tested our approach. Section 3 discusses the pre-processing and the extraction of speech acoustic features. Section 4 elaborates on our proposed approach. The results of our experiments are presented in Section 5. Finally, we close the paper in Section 6.

2 Speech Databases

We tested our proposed approach using the Berlin Database of Emotional Speech and the RML Database.

The **Berlin Database of Emotional Speech**[1] [6], also known as **Berlin EmoDB**, has been utilized in several studies as a benchmark database to test the performance of a technique for speech emotion recognition. The database was constructed using 10 speakers, 5 males and 5 females, who read 10 sentences in German that have little emotional content textually but they are read in such a way to simulate emotions. Originally there were seven discrete emotions considered. The sentence utterances are of variable lengths ranging from 1 to 4 seconds. There is a total of 535 sentence utterances that were evaluated and labelled in a perception test. We only retained those utterances which have emotion labels "Anger", "Disgust", and "Sadness". The recording was done in a studio with high-quality recording apparatus and stored in a mono wave file format with sample rate of 16,000 Hz and quantitative bits of 16.

The **RML Emotion Database**[2] contains 720 audio-visual emotional expression samples that were collected at Ryerson Multimedia Lab. Among the emotions that were expressed we only used "Anger", "Disgust", and "Sadness". This database is language and cultural background independent. The video samples were collected from eight human subjects, speaking six different languages (English, Mandarin, Urdu, Punjabi, Persian, Italian). Different accents of English and Chinese were also included. The samples were recorded at a sampling rate of 22050 Hz using a single channel 16-bit digitization. Samples have length ranging from 3 to 6 seconds with emotions being expressed.

For the two databases we consider sentence utterances as our speech signals.

3 Pre-processing and Feature Extraction

Instead of dealing with sequence of values and taking into account the dynamic nature of speech signal we took a slightly different approach. We firstly cut each speech signal into segments of 250 milliseconds (ms) each and we represent each segment by a single static feature vector. We found out that a length of 250 ms achieves good trade-off between emotional content and variability. In this part, features do not directly describe the whole speech signal but rather they describe the individual segments, thus a speech segment becomes our unit of analysis.

[1] `http://database.syntheticspeech.de/`
[2] `http://www.rml.ryerson.ca/rml-emotion-database.html`

3.1 Pre-processing

The speech signals were initially preprocessed by removing the silent parts at the beginning and end of the signals. Then they were cut into non-overlapping segments of 250 ms each. Here, we did a blind segmentation approach where no prior delimitation of word or syllable boundary has been performed. We assumed that segments may contain emotional primitives that contribute to the over-all emotional content of a speech signal. The segmentation as well as the subsequent extraction of features are illustrated in Figure 1

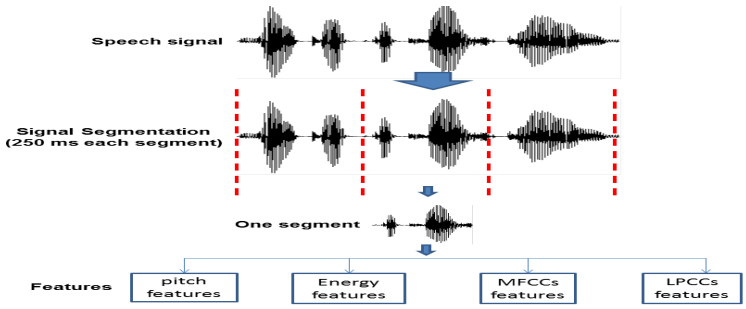


Fig. 1. Segmentation and Feature Extraction Part. An utterance is segmented into non-overlapping speech segments of 250 ms each. The features at this point are extracted from the individual speech segments.

3.2 Speech Acoustic Features

Here the unit of analysis is not the utterance but the 250 ms speech segments cut from it. To extract the segment-level features we apply short time analysis. From the name itself, short time analysis extract short term features on a frame basis; the reason why this type of analysis is also called frame-based analysis. We start by first decomposing the segments into a series of overlapping frames of specified duration. Here the duration of each frame is 25 ms and obtained every 10 ms using a Hamming window function. Four groups of feature types were considered: pitch, energy, mel-frequency cepstrum coefficients, and linear prediction cepstrum coefficients. These acoustic features have been demonstrated to be useful indicators of some emotions in speech signals.

The tension of the vocal folds and the sub glottal air pressure partially reflect the type of emotion expressed in speech. Pitch signal is produced from the vibration of the vocal folds and its vibration rate is called the fundamental frequency of the phonation F0 or pitch frequency. A lot of algorithms exist to estimate the pitch frequency. In this study we used the algorithm based on the autocorrelation of center-clipped frames. For each frame we computed the

value of pitch frequency and statistics of pitch are obtained for the entire segment. The computed statistics are the minimum, maximum, median, lower- and upper- hinges.

Energy is a basic feature in speech signal and it is related to the arousal level of emotions. We extracted the short-term energy on each frame. For the whole segment we computed the minimum, maximum, median, lower- and upper-hinges of the energies as features.

The **mel-frequency cepstrum coefficients** (MFCCs) are based on the characteristics of the human ear's hearing, which uses a non-linear frequency unit to simulate the human auditory system. It is the most widely used spectral representation of speech in many applications. Among their many advantages, MFCCs are simple to calculate, good ability of distinction, and anti-noise. For each of the frames, twelve standard MFCCs are calculated by following the steps: (1) taking the absolute value of the short time Fourier transform (STFT), (2) warping it to a Mel frequency scale, (3) taking the logarithms at each of the mel frequencies (4) taking the discrete cosine transform (DCT) of the log-Mel spectrum and (5) returning the first twelve components

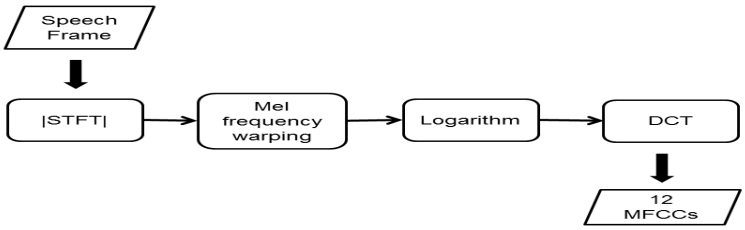


Fig. 2. Diagrammatic representation of the extraction of MFCCs

Figure 2 illustrate the whole process of obtaining the MFCCs. To get the MFCCs features for the whole segment we computed the mean of each of the twelve coefficients from among the constituent frames.

Lastly, **linear prediction cepstrum coefficients** (LPCCs) embody the characteristics of particular channel of speech. The same person expressing different emotions in speech will have different channel characteristics, thus, LPCCs are good to identify emotional content of speech. They are computed from linear prediction coding (LPC) coefficients using a recursive algorithm. As our features we extracted the cepstrum coefficients from 40 LPC coefficients from which we obtained 41 coefficients. The mean of each of the 41 coefficients from frames are calculated to represent the entire segment.

We refer the reader to [7] for additional information regarding the extraction of the preceding acoustic features. A total of 63 features (5 pitch related statistics, 5 energy related statistics, 12 MFCCs and 41 LPCCs) were considered in this study. The features discussed here are used in the first step of our proposed approach.

4 Our Proposal

Our proposed approach does not only achieve maximum detection rate for the three types of emotion but also it provides added information toward the understanding of the synergy among emotions within a speech signal. By doing this, we are able to make a detailed analysis of a speech signal by examining its speech segment components. The characterisation of speech segments could give us a complete emotional expression of the whole signal. In other words, by examining the parts we understand the whole.

Our proposal consisted of two steps. The whole process is shown in Figure 3. The first step involved building models for each emotion class. Also included in this step is the computation of distance measures which will quantify the deviations of the speech frames to each emotion class. The second step will involved the construction of the second set of features by computing summary statistics of the distance values and the training of speech signal classifier.

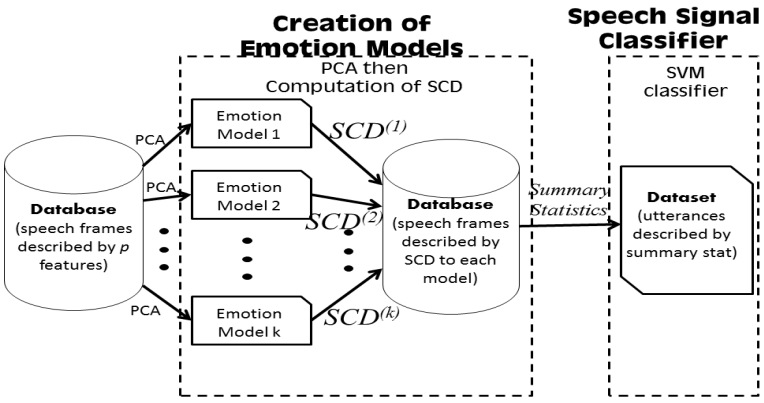


Fig. 3. Diagrammatic presentation of our proposed approach

4.1 Preliminaries

Let us denote an emotion category (or emotion model) as E^j, for 3 emotions we have $j = 1, 2, 3$. Remember that a speech signal (mother signal) is cut into speech segments (or simply segments) of 250 ms duration. We do this because in actual conversation speech signals come in continuous form and not per utterance with clear boundaries. Furthermore, this course of action saves us from the unnecessary computational step of identifying word or syllable boundaries to compute the features. This makes our approach more practical since the scheme is suitable for real-time processing and adaptable to stream analysis.

Depending on the length of the speech signal the number of segments may vary. We assigned emotion labels for each of the segments during training. Here we assume that *the emotion label of a segment is the same as the emotion label of the mother signal where that segment came from.* We represent a segment as $\mathbf{s}_i^j$, which can be interpreted as segment i that has emotion label E^j. Finally each segment is described by p (=63) features. Thus for a given segment its static single feature vector representation is $\mathbf{s}_i^j = (s_{i1}^j, s_{i2}^j, \ldots, s_{ip}^j)^T$. This part is a good reference if ever you need to refresh your memory regarding certain notations.

4.2 Construction of Emotion Models

The motivation for this step is to reveal underlying structure for each emotion categories. This way we understand better the characteristics of speech frames in each emotion categories. Also, this stage will involve a feature reduction part since we want to derive features that are really useful to describe each group.

To obtain the emotion models, we run Principal Component Analysis (PCA) on each emotion classes. After we run PCA on each group E^j we will obtain a matrix of scores T^j and loadings P^j for each group. Since we run separate PCA on each emotion classes we can now summarize each emotion classes in different subspace models (according to the PCA models). The number of retained principal components for each class is denoted by $k_j \ll p$. The relevance of the features on each model can be assessed by examining the loadings of the features on the extracted principal components.

Once we have the emotion models, we can now compute the deviations of each segments to the different models. We can define two deviations: the *orthogonal distance* (OD) and the *score distance* (SD).

The orthogonal distance is simply the Euclidean distance of a segment to an emotion model in its PCA subspace. To compute the orthogonal distance of any segment $\mathbf{s}$, we first compute its projection $\mathbf{s}^{(j)}$ on emotion model E^j. Its projection is given by

$$\mathbf{s}^{(j)} = \bar{\mathbf{s}}^j + P^j (P^j)^T (\mathbf{s} - \bar{\mathbf{s}}^j) \tag{1}$$

where $\bar{\mathbf{s}}^j$ is the feature vector mean (column means) of the speech frames in group E^j. The OD to group E^j of the segment $\mathbf{s}$ is defined as the norm of its deviation from its projection, specifically,

$$\mathrm{OD}^{(j)} = \|\mathbf{s} - \mathbf{s}^{(j)}\| \tag{2}$$

On the other hand, the score distance is a robust version of the Mahalanobis distance measured in PCA subspace. Hence, the score distance of $\mathbf{s}$ is provided by:

$$\mathrm{SD}^{(j)} = \sqrt{(\mathbf{t}^{(j)})^T J^{-1} \mathbf{t}^{(j)}} = \sqrt{\sum_{a=1}^{k_j} \frac{(t_a^{(j)})^2}{\lambda_a^{(j)}}} \tag{3}$$

where $\mathbf{t}^{(j)} = (P^j)^T(\mathbf{s} - \bar{\mathbf{s}}^j) = (t_1^{(j)}, t_2^{(j)}, \ldots, t_{k_j}^{(j)})^T$ is the score of $\mathbf{s}$ with respect to the E^j group, $\lambda_a^{(j)}$ for $a = 1, 2, \ldots, k_j$ stands for the largest eigenvalues in the E^j group, and J is the diagonal matrix of the eigenvalues. The advantage of SD over OD is that SD uses information about the eigenvalues.

In the usual case we need to decide which of the two distances is more appropriate for the problem, but we opted to combine the two distances by using a parameter γ. Specifically we define the combined distance and named it *scortho distance* (SCD) for given $\mathbf{s}$ by

$$\text{SCD}^{(j)} = (\gamma)\text{OD}^{(j)} + (1 - \gamma)\text{SD}^{(j)} \tag{4}$$

where $\gamma \in [0, 1]$. We can optimized the results by choosing appropriate values for the new parameter. One way to choose the parameter is to perform cross-validation and optimizing certain criterion like test prediction accuracy.

Another advantage of building emotion models is we can identify segments which are markedly far from any of the models. We do this by identifying a cut-off value in the computed distances. The cut-off values can be computed by examining the distribution of the distance values. For example, in the case for the score distance, the squared score distances follow asymptotically a χ^2-distribution with k_j degrees of freedom thus we can set $c_{SD}^j = \sqrt{\chi^2_{k_j;0.975}}$ as the cut-off value. We can perform the same kind of analysis for the orthogonal distance and the scortho distance. Segment which computed distances are outside the cut-off values for all the models with respect to a particular distance are termed unrepresentative segments because purportedly they do not contain any emotion. Another possibility is they may form an unknown group and after detection can lead to a new knowledge about the nature of the problem we are studying. In this paper, we are not yet going to pursue the idea of identifying unrepresentative segments.

To summarize the first step of our proposed approach, we first build emotion models using PCA and then proceed to the computation of the SCD. At this point, each speech frame is represented by a vector consisting of its distances to each emotion models. Thus, in our case since we have three emotions, the speech frames are now represented by a vector of three components. Notationally, using the SCD distances, given $\mathbf{s}$, we have

$$\mathbf{s} = (\text{SCD}^{(1)}, \text{SCD}^{(2)}, \text{SCD}^{(3)}) \tag{5}$$

This new representation of the speech frames will be used in the second step.

4.3 Speech Signal Level Classifier

The new representation of the segments obtained from the previous step is aggregated in the speech signal level. Remember that we cut the speech signals into segments so that we can proceed with the initial feature extraction. The aggregation is made possible by computing some summary statistics. Suppose we have a speech signal u and the speech frames cut from it are $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m\}$.

One summary statistic that we can compute is the mean. Hence, using the mean we can represent the speech signal u as

$$\mathbf{u} = (\mu_1, \mu_2, \ldots, \mu_l) \tag{6}$$

where l is the number of emotion categories and

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} \mathrm{SCD}_i^{(j)} \tag{7}$$

where, $\mathrm{SCD}_i^{(j)}$ is the distance of ith segment extracted from u to emotion model E^j. The interpretation of the components is straightforward in this case: the μ_j are the mean SCD of the speech frames components in u to emotion model E^j. We can also view this as the "distance" of our speech signal to each emotion categories.

Aside from the mean, we can also use additional summary statistics like the standard deviation or the inter-quartile range to capture additional information. It is important to note that if we use many statistical measures we will be increasing the size of the vector representation of the speech signals. For instance in our case with three emotions, if we consider two summary statistics then the number of vector components will be equal to 6. Thus it is imperative to choose the right summary statistics to better capture the important characteristics of the speech signals as expressed by the deviations of its member speech segments.

Once we have represented each speech signal u as feature vector $\mathbf{u}$ in the manner we described above we can now construct a speech signal matrix $\mathbf{U}$ denoted by

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_n \end{pmatrix} \tag{8}$$

where n is the total number of utterances or speech signals.

With emotion labels associated to each utterance we are now ready to train the classifier at this step. The user has the liberty to select the classification algorithm he will use here. For our case, we decided to employ Support Vector Machines since it has shown competitive performance in other pattern recognition problems [8, 9]. For an in-depth discussion about the principles of SVM we refer the reader to [10].

5 Results

All throughout the experiments we made use of the features in the speech frame level, namely, 5 pitch related statistics, 5 energy related statistics, 12 *MFCCs*, and 41 *LPCCs* in the first step. In the second step we have aggregated the distances obtained from the first step by computing the median, standard deviation,

and the inter-quartile range of the SCD distances of the speech frames within an utterance with respect to each emotion models. Thus, each speech signal is represented by a vector of 9 elements. Morever, we trained SVM classifier in the second step using the ordinary dot product kernel function. As a standard practice to get reliable estimate of the performance of our proposed approach we use 10-fold cross validation and computed the accuracies and macro F-measure using the test data for each fold. It is important to emphasize here that we trained the models without taking into account the gender or age or language of speakers and the results reported are the mean accuracies and mean macro F-measures on the speech signal level. The whole process of train and testing is depicted in Figure 4. We applied our approach on the two databases separately.

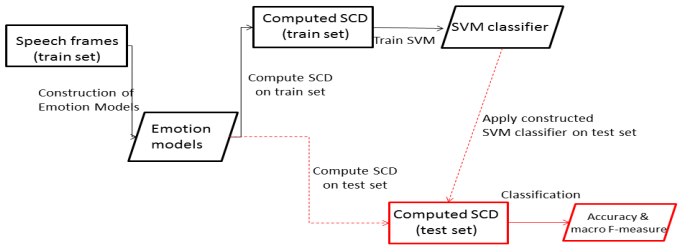


Fig. 4. Training and testing followed in the experiments

Table 1. Average precision and recall for each emotion categories in each database using our proposed approach

Database	Emotion	Ave. Precision	Ave. Recall
Berlin EmoDB	Disgust	0.907	0.800
	Sadness	0.967	1.000
	Anger	0.955	0.95
RML	Disgust	0.850	0.900
	Sadness	1.000	1.000
	Anger	0.975	0.95

From the results shown in Table 1 we find that our approach is able to effectively identify the three types of emotions. Particularly our approach is superior in detecting emotions “Sadness” and “Anger”. This can be explained by the fact that the two emotions have contrasting arousal characteristics as expressed in the speech signal. “Sadness” is a lowly active emotion and “Anger” is a highly-active one. The case of emotion “Disgust” is interesting because

although it has almost the same arousal as "Anger", the classifier has successfully distinguish it from "Anger" most of the time especially in the RML database.

We present in Table 2 the mean accuracy and mean macro F-measures obtained by our approach. The table confirms our claim that our method is particularly effective in the detection of these three emotions.

Table 2. Performance of our proposed approach on the two databases assessed using average accuracy and average macro F-measure. We also present the baseline accuracy obtained by classifying all speech signals to only one emotion.

Database	Baseline Accu.	Ave. Accu.	Ave. macro F-measure
Berlin EmoDB	33.33%	92.10%	94.10%
RML	33.33%	95.70%	94.60%

Aside from the good detection rates we also get additional knowledge regarding the characteristics of the individual emotion classes. The information is derived from the first step of our methodology. We present in Table 3 the summary of the attributes of the emotion models constructed using the Berlin EmoDB. We find that although we have used an initial 63 features we discovered that we can construct at most 10 (latent) features and still capture the over-all variation in each emotion models. An analysis on the loadings of the original features to the constructed features revealed that energy related features and MFCCs are influential in the detection of "Disgust" and "Sadness" whereas pitch related features, energy related features, and MFCCs are important features for the detection of "Angry". LPCCs seem to have less contribution in this case. Maybe, LPCCs will become necessary if we wish detect other types of emotions.

Table 3. Summary characteristics of each emotion models for the Berlin EmoDB.

	Disgust Model	Sadness Model	Angry Model
PCs retained (latent features)	10	8	10
Explained Variance	100%	100%	100%
Most Important Variables	energy, MFCCs	energy, MFCCs	pitch, energy, MFCCs

6 Summary and Conclusion

The methodology we have proposed in this paper has shown competitive performance in the detection of three emotion types on the two databases. Aside from the good classification accuracies the methods also reveal additional knowledge regarding the individual emotion classes. This knowledge is desirable if we want

to understand better the type of features useful for the discrimination of emotion classes and also the synergy among emotion categories within speech signals.

Another confirmation we got in this study is the usefulness of using speech segments as unit of our analysis. The speech segments acts as atoms of emotions from which we can identify emotional primitives that could be used to deduce the over-all emotion of a speech signal.

In practice, our proposed approach can be implemented relatively fast. In the first step PCA can be run rapidly and the computation of distances is speedy. In the second step the complexity only depends on the complexity of the classifier. Lastly, our proposed approach offer new insights to the kind of analysis that can be done and additional tool to analyse emotional speech database.

In our future work we will investigate further the use of other features both in the first and second steps and classifiers in the second step. We will also test our method on other speech databases and with more number of emotions.

References

1. Cowie, R., Douglas-Cowie, E., Tsapatsoulis, N., Votsis, G.N., Kollias, S.D., Fellenz, W.A., Taylor, J.G.: Emotion recognition in human-computer interaction. IEEE Signal Processing Magazine **18** (2001) 32–80
2. Vogt, T., André, E., Wagner, J.: Automatic recognition of emotions from speech: A review of the literature and recommendations for practical realisation. In Peter, C., Beale, R., eds.: Affect and Emotion in Human-Computer Interaction, From Theory to Applications. Volume 4868 of LNCS. Springer (2008) 75–91
3. El Ayadi, M.M.H., Kamel, M.S., Karray, F.: Survey on speech emotion recognition: Features, classification schemes, and databases. Pattern Recognition **44** (2011) 572–587
4. Koolagudi, S.G., Rao, K.S.: Emotion recognition from speech: a review. International Journal of Speech Technology **15** (2012) 99–117
5. Branden, K.V., Hubert, M.: Robust classification in high dimensions based on the SIMCA method. Chemometrics and Intelligent Laboratory Systems **79** (2005) 10–21
6. Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W.F., Weiss, B.: A database of german emotional speech. In: INTERSPEECH 2005, ISCA (2005) 1517–1520
7. Rabiner, L., Schafer, R.: Introduction to Digital Speech Processing. Foundations and Trends in Technology. Now Publishers (2007)
8. Osuna, E., Freund, R., Girosi, F.: Training support vector machines: an application to face detection. In: CVPR '97, IEEE Computer Society (1997) 130–136
9. Joachims, T.: Text categorization with suport vector machines: Learning with many relevant features. In Nedellec, C., Rouveirol, C., eds.: Proc. 10th European Conference on Machine Learning. Volume 1398 of LNCS., Springer (1998) 137–142
10. Herbrich, R.: Learning Kernel Classifiers: Theory and Algorithms. MIT Press, Cambridge, MA, USA (2001)

Towards extracting relations from unstructured data through natural language semantics

Diana Trandabăț[1,2]

[1] Faculty of Computer Science, University "Al. I. Cuza" of Iasi, Romania
[2] Institute of Computer Science, Romanian Academy
dtrandabat@info.uaic.ro

Abstract. Semantics has always been considered the hidden treasure of texts, accessible only to humans. Artificial intelligence struggles to enrich machines with human features, therefore accessing this treasure and sharing it with computers is one of the main challenges that the natural language domain faces nowadays. This paper represents a further step in this direction, by proposing an automatic approach to extract information from texts on the web by using semantic role labeling.

Keywords: artificial intelligence, natural language parsing, semantic roles, machine learning for semantic labeling.

1 Introduction

Attracted by the potential applications, more and more researchers from the artificial intelligence field submerged into the natural language processing domain. Thus, one further step in the human-computer interaction is the use of human languages instead of some pre-defined expressions. In order to teach a computer to understand a human speech, language models need to be specified and created from human knowledge. While still far from decoding political speeches, computer scientists, electrical engineers and linguists have all joined efforts in making the language easier to be learned by machines.

Our approach uses semantic role analysis in order to establish the roles that entities have in different contexts, and what are the temporal, modal or local constraint that determine or restrict an event to take place. A semantic role represents the relationship between a predicate and an argument. Semantic parsing, by identifying and classifying the semantic entities in context and the relations between them, has great potential on its downstream applications, such as text summarization or machine translation.

In this paper we propose a system which, starting from an input entity, extracts web pages found on a Google search for a particular entity, selects the snippets that contain the input entity, and then performs semantic role labeling to extract the relations between the entity and its context. Thus, our system creates a contextual map by

identifying the role an entity plays in different contexts, as well as the roles played by words frequently co-occurring with the input entity.

The motivation behind the work presented in this paper is the need to create a map of structured context related to a specific entity (e.g. a company or product name, an event, etc.). Through this map, the concepts that are usually in relation to the searched input entity are highlighted, together with their specific role (which can be of type Cause, Effect, Location, Time, etc.), thus providing a good material for social analyses, market research or other marketing purposes.

The paper is structured in 5 sections. After an introduction in the field of semantic role analysis, we briefly present the current work in Section 2. Section 3 introduces the overall application, describing the intermediary steps, while section 4 presents our approach for a Semantic Role Labeling system and evaluates it. The final section draws the conclusions of this paper and discusses further envisaged developments.

2 Semantic Role Analysis

Natural language processing is a key component of artificial intelligence. All content elements of a language are seen as predicates, i.e. expressions which designate events, properties of, or relations between, entities. The predication represents the mechanism that allows entities to instantiate properties, actions, attributes and states. More precisely, the linking between a phenomenon and individuals is known as predication. Predicates are not treated as isolated elements, but as structures, named predicate frames or semantic frames. Fillmore in [7] defined six semantic roles: Agent, Instrument, Dative, Factive, Object and Location, also called deep cases. His later work on lexical semantics led to the conviction that a small fixed set of deep case roles was not sufficient to characterize the complementation properties of lexical items, therefore he added Experiencer, Comitative, Location, Path, Source, Goal and Temporal, and then other cases. This ultimately led to the theory of Frame Semantics [6], which later evolved into the FrameNet project [1].

The semantic relations can be exemplified within the Commercial Transaction scenario, whose actors include a buyer, a seller, goods, and money. Among the large set of semantically related predicates, linked to this frame, we can mention buy, sell, pay, spend, cost, and charge, each of which indexes or evokes different aspects of the frame.

In the last decades, hand-tagged corpora that encode such information for the English language were developed (VerbNet [13], FrameNet and PropBank [18]). For other languages, such as German, Spanish, and Japanese, semantic roles resources are being developed. For Romanian, [23] has started to automatically build such a resource.

For role semantics to become relevant for language technology, robust and accurate methods for automatic semantic role assignment are needed. Automatic Labeling of Semantic Roles is defined as identifying frame elements within a sentence and tag them with appropriate semantic roles given a sentence, a target word and its frame [14]. Most general formulation of the Semantic Role Labeling (SRL) problem sup-

posed determining a labeling on (usually but not always contiguous) substrings (phrases) of the sentence s, given a predicate p.

In recent years, a number of studies, such as [3] and [8], have investigated this task on the FrameNet corpus. Role assignment has generally been modeled as a classification task: A statistical model is trained on manually annotated data and later assigns a role label out of a fixed set to every constituent in new, unlabelled sentences. The work on SRL has included a broad spectrum of probabilistic and machine-learning approaches to the task, from probability estimation [8], through decision trees [22] and support vector machines [20], to memory-based learning [15]. While using different statistical frameworks, most studies have largely converged on a common set of features to base their decisions on, namely syntactic information (path from predicate to constituent, phrasal type of constituent) and lexical information (head word of the constituent, predicate).

As for extracting relations, worthnoticing is the recent work done by the Watson group at IBM on relationship extraction and snippets evaluation with applications to question answering [21, 27].

3 Towards Semantically Interpreting Texts

The goal of the application we propose is to extract the context in which an entity occurs in web documents, together with the relations that the searched entity establishes with frequently co-occurring words. The basic architecture of our application contains a series of specialized modules, as follow:

3.1 User Input Acquiring Module

This module prompt the user for an input entity, usually representing a company or product name, an event name, a person, etc. A drop down list of the most frequent searched entities is offered as suggestion.

3.2 Web Page Retrieval Module

This module extract from the web the first n (in our tests n=200) web pages found on a Google search for the input entity. Google search options restricting the web search can be applied, such as selecting articles from newspapers, blogs or in a specific language.

3.3 Snippet Extraction Module

Using the Google snippet suggestion and some simple heuristics, the paragraphs containing the input entity are selected. A simple anaphora resolution method, based on a set of reference rules, is applied to the web pages, in order to link all entities to their referees.

The anaphoric system we used is a basic rule-based one, focusing on named entity anaphoric relations. Thus, we developed a rule-based system that performs the following actions:

- identifies a subset of a named entity with the full named entity, if it appears as such in the same text. For instance, Caesar is identified with Julius Caesar if both entities appear in the same text. Similarly, the President of Romania and the President are considered anaphoric relations of the same entity, if they appear in a narrow word window in the text.
- solves acronyms using a gazetteer we have initially built over the Internet, and which is continuously growing in size. For instance, *United States of America* and *USA* are co-references.
- searches for different addressing modalities and matches the ones that are similar. For instance, *John Smith* is co-referenced with *Mr. Smith*, and *Mary and John Smith* is co-referenced with *The Smiths*, or *The Smith Family*.
- solve pronominal anaphora in a simplistic way. Thus, if a pronoun (i.e. *she*, *he*, *him*, *his* etc.) is found in the text, and in the preceding sentence an entity is found, then we create an anaphoric link between the pronoun and its antecedent. A similar rule exists for companies, where the pronoun *it* may be linked to *the Insurance Company*, for instance.

3.4 Snippet Cleaning Module

After relevant paragraphs (containing the input word) are extracted, functional words such as (and, so, a, etc.) are eliminated from these paragraphs, since, being statistically too frequent for any kind of texts, they do not convey any useful information for semantic role labeling. A list of these functional words, created by using word frequencies in large corpora, is used.

3.5 Semantic Role Labeling Module

This module performs semantic role labeling on the obtained paragraphs, in order to identify the role the entity in question and the related entities plays. It will be described in details in the following section and evaluated in Section 6.

3.6 Module for the Creation of a Map of Concepts

This modules works in two steps: first, it extracts from the semantic role analysis the relations between the searched entity and the neighbor entities, creating a list of relations. Secondly, it generalizes the concepts that are found to be in relation with the searched entity across all extracted paragraphs, using the WordNet [5] hierarchy.

The next section presents the core module of this architecture, the semantic role labeling system, developed by training a set of supervised machine learning algorithms for several languages.

4 Our Semantic Role Labeling Approach

In order to detach semantic information from texts, we built a supervised SRL system, which we named PASRL – Platform for Adjustable Semantic Role Labeling. Several machine learning techniques and feature sets have been tested using the algorithms implemented in the Weka toolkit [26]. We used 12 of the most common machine learning algorithm that are available in Weka, such as decision trees, SVMs, memory-based learners, etc. A full description of the machine learning algorithm from Weka used for PASRL can be found in [25]. Each learning algorithm is trained with a set of features, the performances of the different obtained models are compared, and the best one is selected. The output of PASRL is a Semantic Role Labeling System which can be used to annotate new texts.

Using training data preprocessed with syntactic and dependency information, PASRL is composed of two main sub-systems: a Predicate Prediction module and an Argument Prediction module.

4.1 Predicate Prediction module

The first module of the semantic role labeling system is the predicate prediction module. The Predicate Prediction module system is composed out of three sub-modules:

- *Predicate Identification* this module takes the syntactic analyzed sentence and decides which of its verbs and nouns are predicational (can be predicates), thus for which ones semantic roles need to be identified;
- *Predicate Sense Identification* – once the predicates for a sentence are marked, each predicate need to be disambiguated since, for a given predicate, different sense may demand different types of semantic roles;
- *Joint Predicate and Predicate Sense Identification* – jointly identifies the predicates and their senses (the two above sub-modules).

Predicate Identification Task

Our semantic role labeling system uses the PropBank annotation of semantic roles. Since predicational words are not just verbs, beside PropBank [18] for the verbal frames, NomBank [20] is also used for nouns. Using syntactic annotation, consisting of marked dependency relations, and also the resources PropBank and NomBank, the system first tries to identify the words in the sentence that can behave as semantic predicates, and for which semantic roles need to be found and annotated (the Predicate Identification module). This module relies mainly on the external resources, thus the verbs that are in PropBank (have semantic frame annotation) are likely to be semantic predicates, those which aren't, are not predicational verbs, thus cannot have semantic arguments. For example, the verb to be has no annotation in PropBank, since it is a state and not an action, predicational, verb. Similarly, the NomBank is used to sort nouns that can behave as predicates from those that cannot have semantic arguments.

The predicate identification program transforms the annotated input into training instances for the ML algorithms in order to identify which nouns or verbs from the input are predicates. For each noun or verb in the sentence, an instance is created with a set of features from a syntax-based vector space, inspired from the features usually used for Semantic Role Labeling [14], and a binary class label (the candidate word is or not a predicate for the considered sentence). The features used for the Predicate Prediction are detailed in [24].The output of this module is the input file, where each verb or noun that behaves as a predicate is annotated.

After the predicates from the input sentence are identified, the next module is successively applied for all the predicates found in the sentence, in order to identify for all of them all and only their arguments. For example, for the sentence:

`The assignment of semantic roles depends on the number of predicates.`

two predicates are identified by the Predicate Identification module (assignment and depend). The next module will be applied two times, once for the identification of the sense and semantic arguments of the assignment predicate, and once for the depend predicate.

The output will therefore provide the two annotations:

$[\text{The assignment of semantic roles}]_{\text{ARG0}}$ $[\text{depends}]_{\text{TARGET}}$ $[\text{on the number of predicates}]_{\text{ARG1}}$.
$[\text{The assignment}]_{\text{TARGET}}$ $[\text{of semantic roles}]_{\text{ARG1}}$ depends on the number of predicates.

Predicate Sense Identification Module

Since this module considers that the predicational words are already identified, a binary feature is_predicate is extracted from the training file and added to the feature set created for the Predicate Identification task, and Weka classifiers are again ran to classify each predicate with its PropBank/NomBank role set. This module is needed in order to select the types of semantic roles specific to the sense the predicate has. The sense annotation in PropBank and NomBank is similar to some extent to the sense annotation from WordNet, with the observation that the classification in sense classes (role sets in PropBank's terminology) is centered less on the different meanings of the predicational word, and more on the difference between the sets of semantic roles that two senses may have. The senses and role sets in PropBank for a particular predicate are usually subsumed by WordNet senses, since the latter has a finer sense distinction.

Joint Predicate and Predicate Sense Identification

Instead of running the Predicate Identification and the Predicate Sense Identification processes successively, we tested running them simultaneously, using the same features presented above and as class the predicate role set similar to the one used for Predicate Sense Identification.

4.2 Argument Prediction Module

After running the first module of the semantic role labeling system (either pipelined or joint), the Argument Identification task is called in order to assign to each syntactic constituent of a verb / noun its corresponding semantic role. The instances in this case are not only the nouns and verbs, but every word in the sentence.

The Argument Prediction module performs argument prediction, based on the dependency relations previously annotated with the MaltParser [16] and the Predicate Prediction output. The input of this module contains syntactic information (part of speech and syntactic dependencies), predicate and predicate role set, and in the output each syntactic dependent of the verb is labeled with its corresponding role. This module uses as external resources PropBank and NomBank frame files and a list of frequencies of the assignment of different semantic roles in the training corpus. The features used are described in [24].

5 Evaluating PASRL

An evaluation metric for semantic roles have been proposed within CoNLL shared task on semantic role labeling [14]. The semantic frames are evaluated by reducing them to semantic dependencies from the predicate to all its individual arguments. These dependencies are labeled with the labels of the corresponding arguments. Additionally, a semantic dependency from each predicate to a virtual ROOT node is created. The latter dependencies are labeled with the predicate senses.

The evaluation of the PASRL performance was computed using 10-fold cross-validation on the training set. For each task, PASRL evaluates all the machine learning algorithms used against the gold-annotated corpus, and the best performing algorithm is saved in a configuration file. The evaluation was performed considering the number of correctly classified labels and correctly identified predicates.

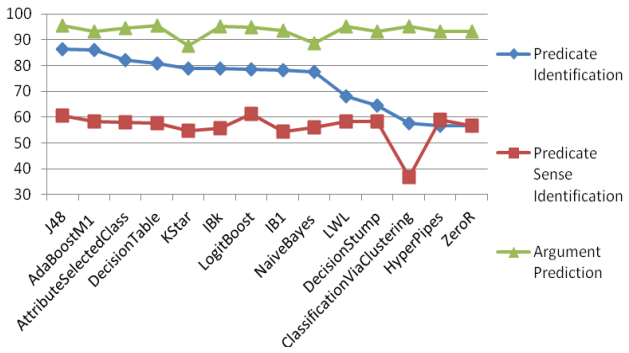


Fig. 1. Performance of the analyzed machine learning algorithms for the SRL tasks performed by PASRL

Fig. 1 presents an overview of the performance of different machine learning algorithms for each of the SRL tasks for English: Predicate Predication (using the Predicate Identification and the Predicate Sense Identification modules, not the joint learning module) and Argument Prediction. One can clearly see that in the Argument Prediction task the algorithms give best results, and the Predicate Sense Identification task is the most difficult task to model. Detailed results for the Predicate Prediction task and its sub-tasks are presented below.

For the Predicate Identification task, running the classifiers with the default weights of Weka for the English dataset, their results ranged from 86% correctly identified predicates to 56%. The algorithm that performs best is the J48 classifier (decision tree) and the one that performs worst is the simple ZeroR classifier (see Table 1).

Table 1. Top 5 ML algorithms evaluated with 10-fold cross-validation for English, for the Predicate Identification task

ML Algorithm	**10-fold cross validation**
J48	86.323
AdaBoostM1	86.016
AttributeSelectedClass	82.169
DecisionTable	80.921
KStar	78.97

Using boosting techniques with J48 as base classifier could improve further the module's performance. Changing the default weights of the classifiers can modify their performances, but we believe that the hierarchy will not change substantially. However, this remains a direction to address in a further work.

The best performing model (J48 in this case, which is a decision tree learning method) is saved and will be used when the Predicate Identification module will be called from the configuration file for annotating an unlabeled text.

The results for the Predicate Sense Identification Task are considerably worse than the ones for Predicate Identification task (see table 2), even if the results report an evaluation performed on a gold annotated input file. Therefore, the actual results are expected to be even worse. However, we notice that J48 is still among the best algorithms, and memory based algorithms generally perform badly on this subtask.

Table 2. Top 5 ML algorithms for the Predicate Sense Identification task for English

ML Algorithm	**10-fold cross validation**
LogitBoost	61.085
J48	60.641
HyperPipes	58.868
AdaBoostM1	58.322
DecisionTable	57.667

Instead of running the Predicate Identification and the Predicate Sense Identification processes successively, we tested running them simultaneously, using the same features presented above. Fig. 2 shows the difference in the performance obtained by using the pipelined Predicate Identification and Predicate Sense Identification module, as compared to the module that jointly learn Predicate and Predicate Sense Identification. One can notice that, although the results are significantly worse for the joint learning task, the algorithms that perform best for the pipelined task have still good performance in the joint task.

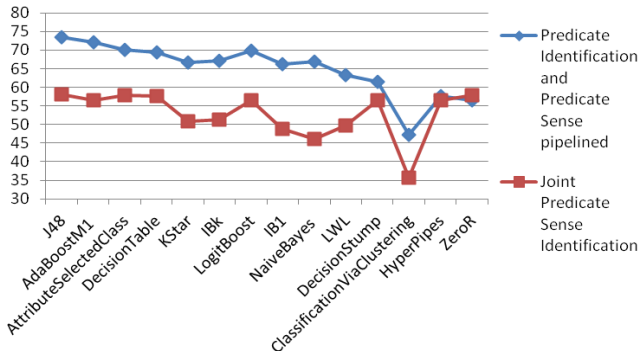


Fig. 2. Performance of the Predicate Prediction system using (1) the pipelined Predicate Identification and Predicate Sense Identification module tans (2) the joint Predicate and Predicate Sense Identification

When considering the 5 best classifiers for all the sub-tasks trained on the whole training data, evaluated using 10-fold cross-validation, we can observe that J48 and Decision Tables are among the 5 best algorithms for different languages, which suggests that some algorithms may perform better than others for Semantic role Labeling.

6 PASRL in Multilingual Context

PASRL was developed using training sets for different languages: English, German, Chinese, Czech and Japanese, provided for research purposes by the CoNLL 2009 shared task. The training data consisting of manually annotated treebanks such as the Penn Treebank for English, the Prague Dependency Treebank for Czech and similar treebanks for Chinese, German and Japanese languages, enriched with semantic relations. This allows for multilingual comparison to be performed. Thus, one can notice that the best performing algorithms score differently for different languages.

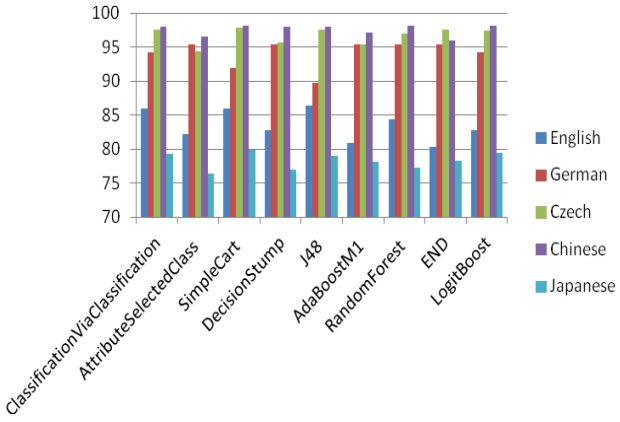


Fig. 3. Performance of the analyzed machine learning algorithms for the Predicate Identification task for different languages

For instance, as figure 3 shows, the best performances for Predicate Identification for Czech, German and Chinese range around 97-98%, while for English the best performance is 86% and for Japanese only almost 80%.

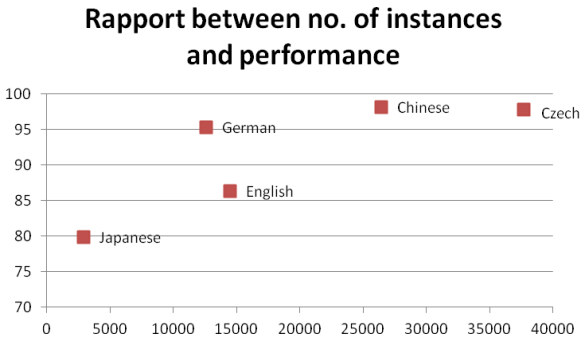


Fig. 4. Rapport between the number of instances and the performance of the Predicate Identification module in different languages

Since the size of the training corpus is similar (see figure 4 for an overview of the number of training instances), one possible explanation could be the different level of inflection, the free vs. fixed word order, or the position of the verb with respect to the other elements of the sentence (SVO vs. SOV language type), or a combination thereof.

7 Conclusions

This paper presented a semantic role labeling system developed using supervised machine learning algorithms from the Weka framework. This system is used in an application that monitors the contexts in which a specific entity appears in web texts and the relations it has with other co-occurring concepts. The developed SRL platform can be used for different languages, provided that a training corpus annotated with semantic roles is available. After testing several classifiers on different subproblems of the SRL task (Predicate Identification, Predicate Sense Identification, Predicate and Sense Identification, Argument Prediction), the proposed system chooses the algorithm with the greatest performance and returns a Semantic Role Labeling System (a sequence of trained models to run on new data).

The envisaged application of our system is in the marketing field, where the related concepts our system offers can be used to monitor the reaction of the consumer to different changes in a company's marketing policies.

Acknowledgments This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS – UEFISCDI, project number PN-II-RU-PD-2011-3-0292.

References

1. Baker G., Collin F., Fillmore, Charles J., and Lowe, John B. "The Berkeley FrameNet project". *In Proceedings of COLING-ACL*, Montreal, Canada. 1998.
2. Budanitsky Alexander and Graeme Hirst. "Evaluating wordnet-based measures of semantic distance". *Computational Linguistics*, 32(1):1347, 2006.
3. Chen J. and O. Rambow. "Use of deep linguistic features for the recognition and labeling of semantic arguments". In *Proceedings of EMNLP 2003*.
4. Daelemans, W., Zavrel, J., K. van der Sloot and A. van den Bosch. *TiMBL: Tilburg Memory Based Learner*, version 5.1, Reference Guide. Technical report, ILK Technical Report Series 04-02, 2003.
5. Fellbaum Christiane (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.
6. Fillmore Charles J. "Frame semantics", în *Linguistics in the Morning Calm*, Hanshin Publishing, Seoul , 1982, 111-137.5
7. Fillmore Charles J. "The case for case". In Bach and Harms, editors, *Universals in Linguistic Theory*, pages 1-88. Holt, Rinehart, and Winston, New York, 1968.
8. Gildea Daniel and Daniel Jurafsky. "Automatic labeling of semantic roles". *Computational Linguistics*, 28(3):245-288, 2002.
9. Hacioglu Kadri and Wayne Ward. "Target word detection and semantic role chunking using support vector machines". *In Proc. of HLT/NAACL-03*, 2003
10. Hacioglu Kadri. "Semantic role labeling using dependency trees". In *Proceedings of the 20th international conference on Computational Linguistics COLING'04*, Morristown, NJ, USA, 2004
11. Klein Dan and Christopher D. Manning. "Accurate unlexicalized parsing". *In Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423-430, 2003.

12. Kudo Taku. *Machine Learning and Data Mining Approaches to Practical Natural Language Processing*. PhD thesis, School of Information Science, Nara Institute of Science and Technology, 2003.
13. Levin B. and M. Rappaport Hovav. *Argument Realization. Research Surveys in Linguistics Series.* Cambridge University Press, Cambridge, UK, 2005.
14. Marquez Lluis, Xavier Carreras, Kenneth C. Litkowski, and Suzanne Stevenson. "Semantic role labeling: An introduction to the Special Issue". *Computational Linguistics*, 34(2):145-159, 2008.
15. Morante Roser, Walter Daelemans, and Vincent Van Asch. "A combined memory-based semantic role labeler of English". In *Proceedings of CoNLL*, pp 208-212, Manchester, UK, 2008.
16. Nivre J. "An efficient algorithm for projective dependency parsing". In *Proc. of the 8th International Workshop on Parsing Technologies* (IWPT 03), pp. 149-160, 2003.
17. Pado Sebastian and Mirella Lapata. "Dependency-based construction of semantic space models". *Computational Linguistics*, 33(2), 2007.
18. Palmer Martha, Daniel Gildea, and Paul Kingsbury. "The proposition bank: An annotated corpus of semantic roles". *Computational Linguistics*, 31(1):71-106, 2005.
19. Pennacchiotti Marco, Diego De Cao, Paolo Marocco, and Roberto Basili. "Towards a Vector Space Model for FrameNet-like Resources". In *Proceedings of LREC'08*, Marrakech, Morocco 2008.
20. Pradhan Sameer, Kadri Hacioglu, Valeri Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. "Support vector learning for semantic argument classification". *Machine Learning Journal*, 60(13):11-39, 2005.
21. Schlaefer,N., J Chu-Carroll, E Nyberg, J Fan, W Zadrozny, D Ferrucci, "Statistical source expansion for question answering", Proceedings of the 20th ACM international conference on Information and and Knowledge Management (CIKM) 2011.
22. Surdeanu M., S. Harabagiu, J. Williams, and P. Aarseth. "Using predicate-argument structures for information extraction". In *Proceedings of ACL2003*, pp 8-15, Tokyo, 2003.
23. Trandabăţ D. "Towards automatic cross-lingual transfer of semantic annotation", in *6e Rencontres Jeunes Chercheurs en Recherche d'Information (RJCRI) 2011*, ISBN 978-2-35768-024-1, pp. 403-408, 16-18 March, Avignon, France, 2011.
24. Trandabăţ D. "Mining Romanian texts for semantic knowledge", in *Proceedings of Intelligent Systems and Design Application Conference*, ISDA2011, Cordoba, Spain, ISSN: 2164-7143, ISBN: 978-1-4577-1676-8, pp. 1062-1066, 2011.
25. Trandabăţ Diana. *Natural language processing using semantic frames*. PhD Thesis, 2010, http://students.info.uaic.ro/
~dtrandabat/thesis.pdf.
26. Witten Ian H. şi Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques* (Second Edition). Morgan Kaufmann, 2005.
27. Chu-Carroll, J., J Fan, N Schlaefer, W Zadrozny 2012 "Textual resource acquisition and engineering" - IBM Journal of Research and Development, vol. 56 3.4: IBM, pp. 4–1, 2012.

A Sliding Window Approach for Discovering Dense Areas in Trajectory Streams

Corrado Loglisci and Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari "Aldo Moro"
via Orabona, 4 - 70126 Bari - Italy
{corrado.loglisci,donato.malerba}@uniba.it

Abstract. The task of discovering dense areas aims at detecting regions with high concentration of moving objects from trajectory data and it plays an important role in the development of systems related to traffic and transport management. The problem has been investigated under two main assumptions: the representation of the trajectories in the euclidean space and the characterization of the dense areas as pre-defined cells. However, the adoption of mobile devices enables the generation of trajectories in a streaming style, which adds another degree of complexity to the problem. This is an issue not yet addressed in the literature. We propose a computational solution aiming at detecting dense areas from trajectory streams in a network. Our proposal adopts a sliding window strategy which enables the discovery of two types of dense areas, one based on spatial closeness, the other one based on temporal proximity. Experiments are conducted on trajectory streams generated by vehicular objects in a real-world network.

1 Introduction

The recent adoption of the mobile and ubiquitous technologies has enabled the generation of massive data about moving objects. The development of advanced applications has greatly benefited from this new source of information. For instance, applications for monitoring fleets of vehicles have been conceived thanks to the possibility to collect positions recorded by GPS-equipped devices installed on the moving vehicles. Typically, the generated data are trajectories corresponding to sequences of latitude and longitude positions each associated with the time-stamp when the object had that position. Analysing trajectories remains a widely investigated research line due to its impact on prominent fields such as intelligent traffic management and urban transportation design.

A challenging problem related to the analysis of the trajectories is the identification of areas (or regions) with high density. An area is dense if the number of moving objects it contains is above some threshold or if it exhibits a concentration of objects greater than expected. Usually, it is characterized by its spatial location and the associated time-interval.

Most of the existing works (e.g.,[3, 4]) follow a database-inspired approach based on density queries. There, the main problem is to effectively compute answers and it is solved through the simplification of the queries or the integration

of human-specified criteria. Common to these works there is the representation of the space in form of a grid which forces the dense areas to be fixed-size cells. In real-world applications, dense areas do not have a pre-defined form and each area can have a size and shape different from the others. This is typical in spaces structured as road networks where the areas have a so irregular form that it would be difficult to discover dense roads in pre-defined cells.

In this paper, we propose an approach to discover dense areas from trajectories of objects moving in a road network. The network-based representation has a two-fold purpose. First, the consideration of some intrinsic features of space (such as, the presence of buildings and infrastructures) which constraint the movements of the objects, while, in the usual euclidean representation the objects can move freely without particular limitation. Second, the possibility of dealing with the problem at a lower level of granularity through the discovery of dense segments (of the roads) which permit to built areas with any shape and size. Strictly connected with the spatial dimension, we have to take into account the temporal component. Indeed, ignoring it can make the spatial location alone not very useful. We propose to consider time both as dimension of analysis, since the movement and speed of the objects are function of time, and as information associated to a dense area, since some roads can be particularly dense in some hours of day only.

In the real-world applications, we cannot forget the streaming activity of the GPS devices to record the repeated movements of the objects which result in trajectories generated as unbounded sequences of positions. This raises new challenges about the storage, acquisition and analysis that make most of the existing works on dense areas difficult to apply and ineffective. Based on these considerations, we argue that handling trajectory streams becomes necessary, especially in urban contexts where technologies enabled to process trajectories and detect dense areas in real-time can be of support in several practical activities.

To face the streaming activity there are basically two alternatives. First, collecting the trajectories according to a *snapshot* setting which models the positions of the objects recorded in a time-stamp. Second, adopting *sliding windows* which collect positions recorded in periods of time and enable the analysis of the data comprised in a window in the meanwhile another window being created. In this work we follow the second approach since the first one requires that the positions are regularly recorded over time which is an assumption that could not be guaranteed in the moving objects. The second approach instead appears appropriate to model a trajectory in its natural conception of sequence of positions in a time-interval. Coherently to what above illustrated, windows turn out to be particularly suitable to take into account time both as physical measure (with velocity and space) in the process of dense area discovery and as annotation in the representation of those areas. In this work, a sliding window approach is adopted to discover two types of dense areas, one based on the spatial closeness, the other one based on the temporal proximity. In particular, overlapping windows are generated to continuously monitor road segments while a technique, which combines an ad-hoc querying mechanism and rule matching, detects those

dense. The rest of the paper is organized as follows. In the Section 2 we overview the related literature. The scientific problem is formalized in the Section 3, while the Section 4 describes the computational solution. Experiments are described in the section 5. Conclusions in the Section 6 close the paper.

2 Related Work

The problem of discovering dense areas was originally faced through a purely spatial dimension and later under a spatio-temporal perspective where moving objects are considered. Two main approaches can be identified: *i)* density-based clustering methods, and *ii)* density querying on grid-based space.

Although dense areas have been investigated with clustering algorithms, the original problem of the clustering is quite different. In these algorithms, dense areas are determined as those geographic regions which have particular spatial properties and which exhibit a particular concentration of objects. Also, they often work on some assumptions about the data distribution, which is anyway difficult to estimate a-priori. Wang et al. [8] study the problem through spatial and multidimensional domains, and the proposed solution permits to generate hierarchical statistical information from spatial data. A static graph representation of the road network has been considered in [9] where three different clustering techniques (partitioning, density-based and hierarchical) work on a network-based distance notion. The dynamic features of the road network have been instead studied in the research line on the moving objects. Chen et al. [1] define an effective clustering approach which works in road networks: the technique first identifies cluster units and then creates different kinds of clusters based on the units by means of a split-and-merge technique. The network features are exploited to reduce the search space and avoid unnecessary computation of network distance. A quite similar method is implemented in [5] through an algorithm which discovers dense areas from cluster units. The latter are generated as groups of moving objects on the basis of the locations and moving patterns. A different perspective is provided in [6] where clustering is performed on sub-trajectories. The idea is that movements can be similar only in segment of the whose trajectories. The approach first partitions the data into line segments, then group them with a density-based clustering.

The grid-based representation for dense areas is not new and it was originally used in [3] in association with the density queries. The original space is modelled in the euclidean framework and partitioned into fixed-size cells to generate cells efficiently. The accuracy loss, introduced by the pre-defined partitioning, is mitigated in the work of Jensen [4].

In the literature of trajectory streams, most of the works are based on clustering. Costa et al [2] propose an effective and accurate solution to find similarities in trajectories modelled with the Fast Fourier Transform. The approach adopts a window-based mechanism finalized to merge newly created clusters with those oldest. A slightly different problem is the grouping of objects moving together [7]. The method first performs a grouping step at each snapshot of the streamed

trajectories, and then completes an intersection operation among the groups previously created. The integration of micro-groups of objects and smart intersection operations make that method efficient. However, in both these research lines no paper focused on the discovery of dense areas can be enumerated, so this work can be considered as the first attempt.

3 Basics and Problem Definition

Before formally defining the problem we intend to solve, some definitions are necessary. Consider $T : \{t_1, t_2, \ldots, t_k, \ldots,\}$ be the discrete time axis, where $t_1, t_2, \ldots, t_k$ are time-points equally spaced, $M : \{o_1, o_2, \ldots, o_i, \ldots, o_m\}$ be the finite set of unique identifiers of the moving objects, $p_k^i \in \mathbb{R}^2$ is the latitude and longitude position of the i^{th} object at the time-point k^{th}. A trajectory stream S_i is the unbounded sequence of positions associated to the object i^{th}, $S_i = \{p_1^i, p_2^i, \ldots, p_k^i, \ldots\}$. The time-points of recording of the positions are not necessarily equally spaced.

A road network is modelled as a directed graph $G(V, E)$, where V is the set of vertices representing road intersections and terminal points, and E is the set of edges representing road segments each connecting two vertices. A road segment is defined by a unique identifier and the latitude and longitude positions of the terminal points p_s, p_e. For simplicity, we assume that the speed of an object in a road segment is constant. Given the positions p_h^i, p_k^i $(t_k > t_h)$, we can compute the speed of an object in a segment. While, given the speed, the position p_h^i and the length of a segment, we can compute the time-point t_k $(t_k > t_h)$ when the object is expected to be out of the segment (afterwards, t_{out}), where p_k^i in t_k denotes the position of the object in correspondence to the terminal point of the segment.

In this work, dense segments and areas are detected from the trajectory streams.

Definition 1 (Dense Segment). *A segment $\sigma \in E$ is dense if the following conditions hold:*

- *there exists a set of objects $D \subseteq M$ s.t. $\forall o_i \in D$: $t_{out}^i < t_k^i$, t_{out}^i is the time in which we expected the object o_i is out from the segment. This condition is true when o_i is still in the segment at the time t_k^i and has position p_k^i;*
- *$|D| > \delta_{min}$, δ_{min} a user-defined minimum density threshold*

A dense segment σ is identified by the tuple: $\langle p_s, p_e, D, t_0, t_{max}\rangle$, where $t_{max} = \arg\max_{o_i \in D} t_{out}^i$ $(t_0 < t_{max})$, so $[t_0, t_{max}]$ establishes the interval in which σ is dense.

Intuitively, a segment is dense when there are objects which have not come out from the segment within the expected time which is specific for each object. This notion is coherent with the queueing process.

A collection of dense segments can form a dense area on the basis of the spatial closeness. It would depict a traffic jam where neighbouring segments exhibit density at the same time (the objects are fully stopped). More formally,

Definition 2 (Spatial Closeness-based Dense Area). *A collection of dense segments Σ is a dense area of type SC if the following conditions hold:*

- $\forall \sigma', \sigma'' \in \Sigma : d(\sigma', \sigma'') \leq L$, *$L$ is a user-defined parameter, d is a network-based distance;*
- $\forall \sigma', \sigma'' \in \Sigma : t'_0 = t''_0$ *(t'_0, t''_0 are the lower bounds the time-intervals of the tuple of σ' and σ'' respectively);*
- $\forall \sigma', \sigma'' \in \Sigma : D' \cap D'' = \oslash$

A collection of dense segments can form a dense area on the basis of the spatial closeness and temporal proximity. It would depict a traffic congestion where near segments exhibit density at consecutive times (the objects flow slowly). More formally,

Definition 3 (Spatial Closeness and Temporal Proximity-based Dense Area). *A collection of dense segments Σ is a dense area of type TP if the following conditions hold:*

- $\forall \sigma', \sigma'' \in \Sigma : t''_0 > t'_0$*;*
- $\forall \sigma', \sigma'' \in \Sigma : t''_0 - t'_0 < \Omega$, *$\Omega$ user-defined parameter;*
- $\forall \sigma', \sigma'' \in \Sigma : D' \cap D'' \neq \oslash$.

The threshold Ω defines the temporal proximity within which the dense areas of type TP can be seek.

Given the set of moving objects M, the trajectory streams in the form of S_i, the road network G, the problem at the hand is *To Detect* dense areas as defined in the Definitions 3 and 4. The parameters δ_{min}, L and Ω are used to filter out meaningless segments and areas.

4 Dense Areas from Trajectory Streams

4.1 Overview

To illustrate the computational solution we need the notion of trajectory window. Let $S_i, S_{i+1}, \ldots S_n$ be a stream of trajectories of objects $o_i, o_{i+1}, \ldots, o_n$. The trajectory window $[W]_w^u$ of width ω is a sub-sequence of the streams $S_i, S_{i+1}, \ldots S_n$ and comprises the positions observed in the time-interval $[t_u, t_w]$, where ω is a user-defined parameter.

The streaming setting requires that the phase of analysis is not postponed to that of acquisition of data. It often precludes the possibility of performing pre-processing operations on the whole dataset which are instead adopted by techniques which do not work on data stream (e.g.[5]). We propose a solution which performs simultaneously a step of data acquisition from trajectories stream and a step of analysis to discover dense areas.

More precisely, the first step collects data (about the positions and segments) continuously coming from moving objects and fills partially overlapping windows.

Each window is overlapped with (some of) the windows that precede and follow it, so it contains part of the positions which are contained in the windows created before and after. For instance, the windows $[W]_w^u, [W]_s^r, [W]_n^m$ are created in this order so that the orders $t_r < t_w$ and $t_m < t_s$ hold, while the order between t_w and t_m cannot be established a-priori since it depends from the rate (defined by the user) with which the windows are generated.

The organization of the windows by partial overlapping allows to share information of the moving objects among different windows and therefore it reduces the possibility of information loss when analyzing the movements.

The second step proceeds at the level of single windows and at the level of sequences of windows. More precisely, it processes one window at time and finds dense areas of type SC from that window: for each segment, the positions buffered in one window are analyzed in order to determine whether that segment is dense. The segments identified as dense in a window will be used to create a dense area of type SC. In the meanwhile one window is processed, the next window is acquired. When the algorithm turns to process the next window, it stores *i)* the dense segments discovered in the past window(s) and *ii)* the expected times t_{out} computed in the past window(s). So, when a new window is processed, the algorithm has the results of the sequence of the windows since there analyzed. The expected times (computed before) are used to complete, in the current window, the search (started in the past window(s)) of dense segments. The discovery of dense areas of type TP is performed on the dense segments obtained from the sequences of windows. Only sequences of windows whose overall width does not exceed Ω are considered.

4.2 Detection of Dense Segments

The method for discovering dense segments resorts to the queueing process according to which the delay of an object (with respect to the expected time) to leave the queue can be attributed to the increase of the number of the objects present in the queue or to a demand for resources greater than expected. We argue that the increase of the moving objects is the most reasonable motivation of the delay in the road networks.

The algorithm combines two activities. The first one performs the monitoring of the segments by means of queries submitted in correspondence of the time-points in which it is expected that the objects will leave the segment. Once a query has been completed, the second activity checks whether the monitored objects are still in the segment: if it is so, they will contribute to make that segment dense. More precisely, the check mechanism adopts heuristics (modelled in the form of *if-then* rules) and performs a matching operation between (the information associated to) each segment and the antecedent parts of the rules. The consequent parts indicate whether the segment is dense or not. The possibility to query the trajectory streams only in correspondence of the expected time-points makes our approach different from the methods in which the queries have to be periodically (and often more frequently) submitted ([4]).

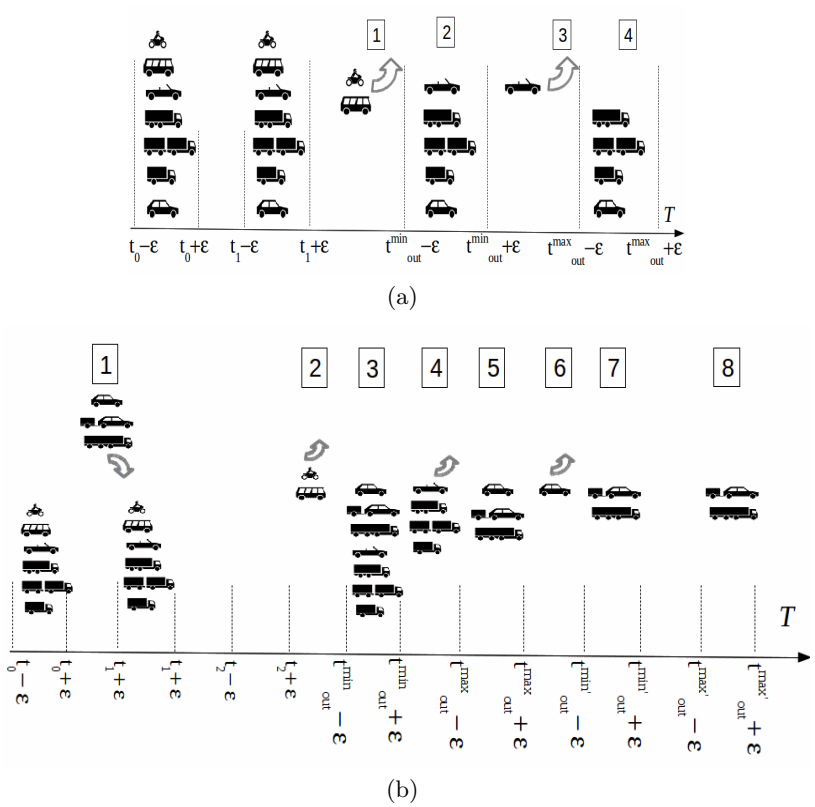


Fig. 1. Estimation of the times t_{out} and application of the check mechanism.

As anticipated in the section 3, the estimation of times requires the speed of each object, its position inside the segment and length of the segment. In particular, the speed of the object is determined when starting to process the current window by submitting two queries (in correspondence of two time-points t_0, t_1) that retrieve the objects to be monitored and their positions, while, the road network G provides the length of the segment. However, the objects can move with different velocities and can leave the segment at different times. This may be determinant in the formation of dense segments and, in order to take into account it, we consider both fast objects and those slow when discovering dense segments. In this sense, we estimate two kinds of expected times, one, denoted as t_{out}^{min}, which indicates the time within which the fast objects should be out, the other one, denoted as t_{out}^{max}, which indicates the time within which the slow objects should be out. More precisely, t_{out}^{min} refers to the slowest object out of the fastest objects, while t_{out}^{max} is the speed of the slowest object in the set of monitored objects of that segment. Intuitively, we expect that the those fastest will be out before (within t_{out}^{min}), while the slowest ones will do it after (but within t_{out}^{max}). If this does not happen, we could have an high concentration

of objects in that segment. We encode this idea into *if-then* rules which work as follows. Once computed t_{out}^{min} and t_{out}^{max}, two queries are submitted (in the same order of t_{out}^{min},t_{out}^{max}) in correspondence of these time-points to check whether the fastest objects (among those monitored) have left before t_{out}^{min} and the slowest objects (among those monitored) have left before t_{out}^{max}: if the number of the objects in common ($|D|$) to the time t_{out}^{min} and to the time t_{out}^{max} is higher than the minimum threshold δ_{min}, then the segment is dense.

Notice that, since it would be unrealistic that the positions of all objects are recorded at the same times, the queries submitted with the value of the time-point could not generate results, so we consider time-intervals with fixed radius ϵ centred on t_{out}^{min} and t_{out}^{max}, and on the first two time-points t_0, t_1. As illustrated in the Figure 1a, the fast objects go out before $t_{out}^{min} + \epsilon$ (square 1), while other fast objects remain, although we expect their absence (square 2). Next, in correspondence of $t_{out}^{max} + \epsilon$ (square 3), we expect that the remaining fast objects and those slow go out: if, even after $t_{out}^{max} + \epsilon$, there are still objects (whose size exceeds δ_{min}), then the segment is dense (square 4).

However, in while identifying the objects to be monitored (queries on t_0 and t_1), new objects enter the segment. The check mechanism could provide wrong results since those objects are not comprised in the initially retrieved set. The solution consists of i) integrating additional *if-then* rules able to recognize the insertion of new objects and ii) adapting the monitoring process since we expect to retrieve the new objects in the next queries. More specifically, an additional query on t_2 ($t_1 < t_2$) is submitted in order to compute the speed and expected time-points (afterwards, $t_{out}^{min'}$ and $t_{out}^{max'}$) for the new objects. The check mechanism is updated in order to monitor the new objects in correspondence of *i)* the expected time-points $t_{out}^{min'}$ and $t_{out}^{max'}$, *ii)* the expected time-points t_{out}^{min} and t_{out}^{max} initially estimated. The values of the new time-points establish the new order of the queries to be submitted and, dependently on this order, different rules will be matched. The time-point t_0 and the longest estimated time-point define (t_{out}^{max} or $t_{out}^{max'}$) the time-interval of the tuple of the segment.

For brevity, here we detail (Figure 1b) only the case in which the time-points $t_{out}^{min'}$, $t_{out}^{max'}$ come after t_{out}^{min} and t_{out}^{max}. As illustrated in the Figure 1b, the presence of new objects in t_1, with respect to t_0, forces to run another query in order to complete the calculation of the speed for the new objects (square 1). In the meanwhile, the algorithm estimates t_{out}^{min} and t_{out}^{max} which will be the time-points that the check mechanism will use for the next queries. After the query t_2, also $t_{out}^{min'}$ and $t_{out}^{max'}$ are estimated, so we have two additional queries to be included in the check mechanism: the new order of the queries will comply with the order of the expected times $t_{out}^{min} < t_{out}^{max} < t_{out}^{min'} < t_{out}^{max'}$. Then, the algorithm checks that the new objects are present in t_{out}^{min} and t_{out}^{max} (squares 2-5), and that all have left before $t_{out}^{max'}$ (square 8). In t_{out}^{min} and t_{out}^{max} we observe that the initially monitored objects leave the segment (squares 2,4) and that other objects remain there (squares 3,5). The latter should leave before $t_{out}^{max'}$, while instead the check mechanism detects that from $t_{out}^{min'}$ to $t_{out}^{max'}$ no object has left (squares 6-8): if these numerically exceed δ_{min}, the segment is dense.

4.3 Detection of Dense Areas

Dense areas are generated with at least two dense segments which meet the Definitions 2 and 3. This provides some hints on the technique to use. The algorithm of the areas of type SC is performed once a window is processed and operates on the segments already detected and dense areas which are being generated during the process. Each segment can be associated to only one area if *i)* its distance from the segments (already added to that area) is lower than the parameter L and *ii)* it has no object in common with those segments. If the examined segment is added to none of the areas, it will be considered as seed for a new area. Finally, we will have a collection of segments which begin to be dense at the time t_0, which is common to all segments of that area.

The dense areas of type TP involve the dense segments obtained from a sequence of overlapping windows. Considering sequences of windows has a two-fold advantage: *i)* mitigating the effect that pre-defined windows (as in the case of areas SC) can have on the final results, and *ii)* enabling the discovery of dense segments on consecutive windows. In particular, when the values of t_{out}^{min} and t_{out}^{max} exceed the last time-point t_w of the window $[W]_w^u$, we need to adapt the check mechanism in order to submit the corresponding queries later, specifically when the window which contains t_{out}^{min} and t_{out}^{max} will be processed. The algorithm acts once a number of windows having a total width less than Ω has been analyzed and it executes two selection operations. The first one takes one segment from each window provided that they are temporally ordered: for instance, given σ',σ'',σ''' detected in the windows $[W]_w^u$,$[W]_s^r$,$[W]_n^m$ respectively, σ',σ'',σ''' are taken if the order $t_0' < t_0'' < t_0'''$ holds. The second one refines the previous operation by selecting the segments which have moving objects in common.

Searching dense segments which share objects over a finite sequence of windows introduces implicitly a spatial neighbourhood in which those objects move around. This allows us to attribute the formation of the congestion areas to specific objects.

For both types of areas, the notion of the distance between two segments σ', σ'' is based on the given network structure and corresponds to the usual distance of the shortest path in the graphs. More precisely, it is the minimum summation of the single distances associated to the intermediate segments and to the segments σ', σ''. Each single distance is the spatial distance between the terminal points of the segment.

5 Experiments

The computational solution was tested on a real-world dataset which comprised trajectories produced by taxies moving in the city of Bejing for one week (02/02/2008-08/02/2008) [1] The original dataset was modified by removing the positions which could not be associated to segments of the network and by adding new data in order to maintain the order of magnitude of the size. Modifications

[1] http://research.microsoft.com/apps/pubs/default.aspx?id=138035

concern the insertions of i) positions in all segments for all objects, ii) positions in the segments of the centre of the city, iii) positions for the objects which have an higher (and lower) number of positions, iv) positions for all objects in peak hours, v) positions in the roads having many segments. Totally, we have almost 37000 roads, 89900 segments, 98600 moving objects. The dataset is poured in a traditional relational DBMS and converted into a data stream by taking the temporal order of the input data as the order of streaming. The road network of the city of Bejing [2] has 141,380 segments and 106,579 vertices from which we derive the lengths of the segments.

Experiments are performed to test the influence of the input threshold δ_{min} on the final dense areas. We report results on the dense segments and dense areas discovered in each day and in the peak hours, namely 7:00-9:00pm, 12:00-14:00pm, 18:00-20:00pm (Figures 2). The thresholds and parameters are set as follows: $\delta_{min} = 3, 5, 7$, $L = 1.5km$, $\Omega = 30mins$, $|t_1 - t_0| = 1min$, $|t_2 - t_1| = 1min$, $\epsilon = 10secs$, $\omega = 5mins$, the rate of generation of the windows equal to $2.5mins$. As to the dense segments, a quite expected behaviour is that the number of dense segments decreases as the minimal threshold increases, and this is common to all days for all hours. An analysis done on the basis of the hours may provide indications of social nature: on the week-end, a greater flow can be observed only in the time-slot 18:00-20:00, while on the weekdays the highest numbers of segments is produced from 12:00-14:00. As to the dense areas, the results basically follow the behaviour of the dense segments. In particular, by analyzing the time-slots 7:00-9:00 and 18:00-20:00, numerous sets of SC-areas are detected when we have more dense segments, while the greatest sets of SC-areas are discovered in correspondence of the highest number of dense segments (12:00-14:00). Also expected it is the result that associates great sets of SC-areas to great sets of segments when δ_{min}=3: in that case, the dense segments can be so numerous to be close to each other with the result to form more dense areas. Numerous collections of TP-areas are obtained when we have many dense segments in all configurations (δ_{min}=3,5,7). Indeed, particularly for the time-slots 12:00-14:00 (on the weekdays) and 18:00-20:00 (on the week-end), the expected high number of objects can justify a slow movement and therefore the raising of TP-areas in all configurations.

We evaluated the final results through a quantitative measure which estimates the capacity of the approach to detect segments which have an high concentration of objects with respect to the neighbourhood. More formally, $\Theta(\sigma) = \frac{\sum_{j=1...m}(\sigma - \sigma_j)}{(m*1)+avg\sigma}$, where σ_j are non-dense segments close to σ in a diameter of 100m[3], m is the number of non-dense close segments, $avg\sigma$ is the average number of objects per segment. The value of $\Theta(\sigma)$ would tend towards 0 with an equal distribution of the objects among the segment σ and its neighbours, while when there is a strong concentration of the objects in the segment σ, the value of $\Theta(\sigma)$ tends towards 1. The Table 1 reports the mean of the values of Θ on all

[2] http://www.openstreetmap.org/

[3] This value has been specifically computed of the road network of the city of Bejing by considering also that two parallel roads can be close within 100m.

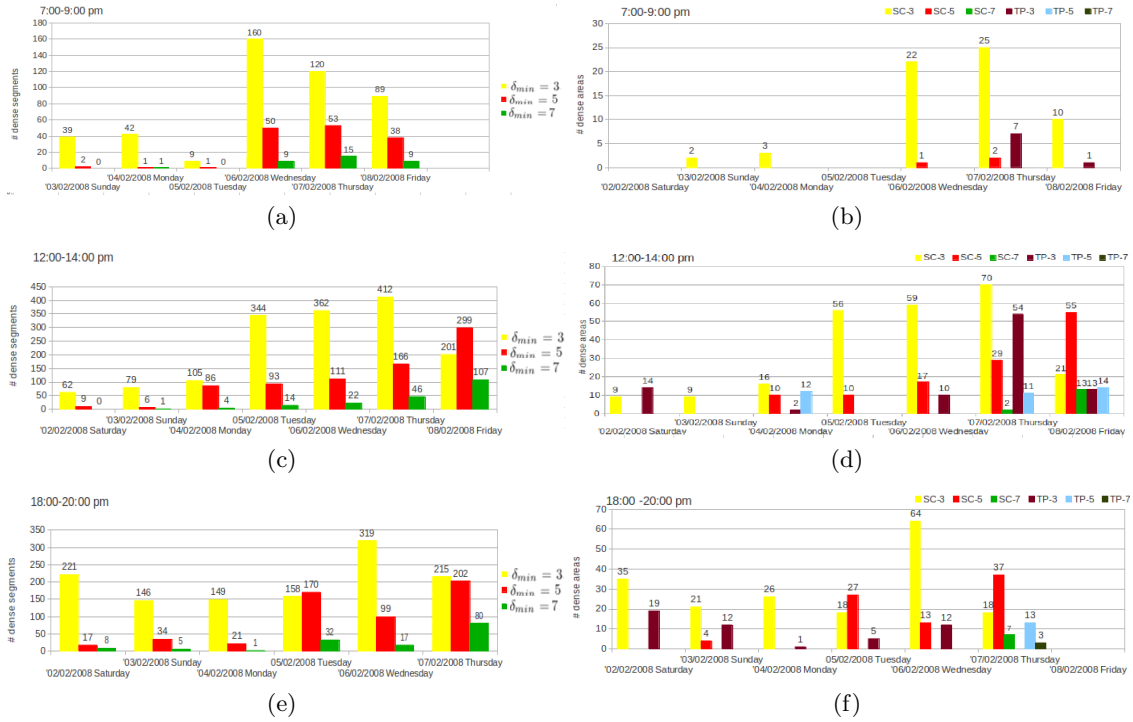


Fig. 2. Dense segments when σ=3,5,7 at peak hours 7:00-9:00pm, 12:00-14:00pm, 18:00-20:00pm (a,c,e). Dense areas of type SC and TP discovered when σ=3(SC-3,TP-3), σ=5(SC-5,TP-5), σ=7(SC-7,TP-7) (b,d,f) .

dense segments discovered in each day: we observe the better performances in the days from Tuesday to Friday. This is encouraging because it points out the ability of the method to recognize correctly high concentrations of objects when the dense segments grow and maintain the robustness with respect to the noise.

6 Conclusions

In this paper we investigated the task of discovering dense areas in a stream of trajectory defined in a road network. The proposed approach adopts a sliding window strategy to detect dense segments and use them to form dense area. It combines knowledge in the form of *if-then* rules and a querying mechanism to retrieve information about the segments from the stream. Experiments prove the applicability to a real-world road network As future direction, we plan to extend the rule base with additional conditions and validate the approach on road networks and trajectory data where ground truth is available.

Table 1. Evaluation on the dense segments by day.

	Θ
02/02/2008 Saturday	0.52
03/02/2008 Sunday	0.67
04/02/2008 Monday	0.41
05/02/2008 Tuesday	0.54
06/02/2008 Wednesday	0.72
07/02/2008 Thursday	0.68
08/02/2008 Friday	0.64

Acknowledgments. This work is in partial fulfilment of the PRIN 2009 Project "Learning Techniques in Relational Domains and Their Applications" funded by the Italian Ministry of University and Research (MIUR).

References

1. J. Chen, C. Lai, X. Meng, J. Xu, and H. Hu. Clustering moving objects in spatial networks. In K. Ramamohanarao, P. R. Krishna, M. K. Mohania, and E. Nantajeewarawat, editors, *DASFAA*, volume 4443 of *Lecture Notes in Computer Science*, pages 611–623. Springer, 2007.
2. G. Costa, G. Manco, and E. Masciari. Effectively grouping trajectory streams. In A. Appice, M. Ceci, C. Loglisci, G. Manco, E. Masciari, and Z. W. Ras, editors, *NFMCP*, volume 7765 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2012.
3. M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-line discovery of dense areas in spatio-temporal databases. In T. Hadzilacos, Y. Manolopoulos, J. F. Roddick, and Y. Theodoridis, editors, *SSTD*, volume 2750 of *Lecture Notes in Computer Science*, pages 306–324. Springer, 2003.
4. C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective density queries on continuouslymoving objects. In L. Liu, A. Reuter, K.-Y. Whang, and J. Zhang, editors, *ICDE*, page 71. IEEE Computer Society, 2006.
5. C. Lai, L. Wang, J. Chen, X. Meng, and K. Zeitouni. Effective density queries for moving objects in road networks. In G. Dong, X. Lin, W. Wang, Y. Yang, and J. X. Yu, editors, *APWeb/WAIM*, volume 4505 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2007.
6. J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 593–604. ACM, 2007.
7. L. A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C.-C. Hung, and W.-C. Peng. On discovery of traveling companions from streaming trajectories. In A. Kementsietsidis and M. A. V. Salles, editors, *ICDE*, pages 186–197, 2012.
8. W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB*, pages 186–195. Morgan Kaufmann, 1997.
9. M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In G. Weikum, A. C. König, and S. Deßloch, editors, *SIGMOD Conference*, pages 443–454. ACM, 2004.

Author Index