Learning Recursive Patterns for Biomedical Information Extraction

Margherita Berardi and Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari
via Orabona, 4 - 70126 Bari - Italy
{berardi,malerba}@di.uniba.it

Abstract. Information in text form remains a greatly unexploited source of biological information. Information Extraction (IE) techniques are necessary to map this information into structured representations that allow facts relating domain-relevant entities to be automatically recognized. In biomedical IE tasks, extracting patterns that model implicit relations among entities is particularly important since biological systems intrinsically involve interactions among several entities. In this paper, we resort to an Inductive Logic Programming (ILP) approach for the discovery of mutual recursive patterns from text. Mutual recursion allows dependencies among entities to be explored in data and extraction models to be applied in a context-sensitive mode. In particular, IE models are discovered in form of classification rules encoding the conditions to fill a pre-defined information template. An application to a real-world dataset composed by publications selected to support biologists in the task of automatic annotation of a genomic database is reported.

1 Introduction

The last decade has witnessed an unexampled expansion of biomedical data and related literature. Advances of genome sequencing techniques have led to an overwhelming increase in the number of publications about discovered genes, proteins and their roles in biological processes. The ability to survey this literature and extract relevant pieces of information is crucial for researchers in biomedicine. However, finding explicit entities (e.g., a protein or a kinase) and facts (e.g., phosphorylation and interaction relationships) in unstructured text is a time consuming and boring task because of the size of available resources, data sparseness and continuous updating of published material. Information Extraction (IE) is the process of mapping unstructured text into structured form, such as knowledge bases or databases, by filling predefined templates of information describing objects of interest and facts about them. This motivates the interest of IE and text mining practitioners toward the biomedical field [24,15].

In a machine learning perspective, IE can be tackled as a classification task, where classification models composed by rules or patterns encoding the conditions to fill a given slot of a template of interest are learned from a set of annotated texts (i.e., examples of filled templates) [21]. Natural language research has widely made use of statistical techniques (e.g., hidden Markov models and probabilistic context-free grammars) because of their robustness and wide coverage peculiarities. However, these techniques cannot properly cope with the level of semantic interpretation. Moreover, they discover linguistically impoverished models which are difficult to interpret and extend [20]. To solve these problems, logic-based approaches, such as those developed in ILP, can be employed. Indeed, they make encoding easy for natural language statements reported both in training data and in the background knowledge [10], and they learn logical theories that can be easily interpreted and revised [10]. Moreover, IE tasks can be naturally framed in the ILP relational setting where data have a relational structure and examples can be related to each other.

Several papers on ILP approaches to learning rule-based models from logical representations of texts are reported in the literature [1,11,16,13,4]. However, only some of them face problems of IE from biomedical texts [5,14], despite the fact that biomedical IE is considered a major application area where ILP may converge [7]. Difficulties are due to the complexity of the biomedical language which is characterized by inconsistent naming conventions, i.e. ambiguities occurring when the same term denotes more than one semantic class (e.g., p53 is used to specify both a gene and a protein) or when many terms lead to the same semantic class (abbreviations, acronym variations). Further problems derive from the continuous creation of new biological terms or evolutions of the same biological object (e.g., genes are renamed once their function is known). The use of non standard grammatical structures as well as domain-specific jargon represent another source of complexity. All these issues make the preparation of training data really difficult. On the other hand, a number of controlled vocabularies, lexicons and ontologies which can be exploited both in the data processing and reasoning steps are available. This further motivates an ILP approach which can naturally handle such a background knowledge.

In the IE literature, there are two main tasks, namely *named entity recognition* and *multi-slot extraction* (or *template filling*). The former aims to identify peculiar objects of interest (the *named entities*), such as the pathology associated to a mutation or the substitution that causes a mutation. The latter looks for conceptual relationships between named entities, such as the genetic mutation associated to both a pathology and a substitution (template $mutation(\langle pathology\rangle, \langle substitution\rangle)$).

A multi-slot extraction task, which is generally based on the results of a named entity recognition task, can be simplified if tagging of named entities is, in its turn, performed by considering some conceptual dependencies implicitly defined at either the syntactic or structural level (e.g., the type of mutation is normally reported after the corresponding substitution). These conceptual dependencies are particularly evident in biomedical domain, since biological systems intrinsically involve relations among several entities (e.g., genes and proteins interacting in regulation networks). Therefore, in this paper, we propose to learn tagging models in the form of *recursive logical theories* which can naturally represent conceptual dependencies between named entities.

The paper is organized as follows. In the following section, we describe the biomedical information extraction problem employed as case study in this work, namely the annotation of a genomic database. Both the data preprocessing techniques and the representation employed for training examples and background knowledge are reported in Section 3. In Section 4, the ILP learning algorithm used to learn recursive logical theories is briefly described. Results obtained on a real-world dataset composed by publications concerning studies on mitochondrial pathologies are reported in Section 5. Finally, some conclusions are drawn in Section 6.

2 The Information Extraction Problem

The application we are addressing concerns the annotation of some resources stored in HmtDB[1], a database of variability and clinical data associated to mitochondrial pathological phenotypes [2]. Currently, HmtDB stores data from healthy subjects while variability and clinical data are manually extracted from published literature. A peculiarity of this fragment of the scientific literature is that biomedical documents are organized according to a regular section structure (composed by Abstract, Introduction, Methods, Results and Discussion) and that often biologists already know which part of the documents may contain a certain kind of information. This suggests to conduct the IE process in a local way to pre-categorized sections of interest [3]. Indeed, selecting relevant portions of text is a prerequisite step for IE, since the lack of robustness and data sparseness makes IE methods inapplicable to large corpora and irrelevant documents. In this application, selected publications concern mitochondrial mutations and biologists are interested in automating the identification of occurrences of specific biological objects (i.e., mitochondrial mutations) and their features (i.e., type, position, involved nucleotides, expressing locus, related pathology) as well as the particular method and experimental setting (i.e., dimension, age, sex, nationality of the sample) reported in the publication.

Let us consider the following example of a text fragment of the collection:

```
Cytoplasts from two unrelated patients with MELAS (mitochondrial
myopathy, encephalopathy, lactic acidosis, and strokelike episodes)
harboring an A-*G transition at nucleotide position 3243 in the
tRNALeU(UUR) gene of the mitochondrial genome were fused with human
cells lacking endogenous mitochondrial DNA (mtDNA)
```

Here `MELAS` is an instance of the *pathology* associated to the mutation under study, `A-*G` is an instance of the *substitution* that causes the mutation, `transition` is the *type* of the mutation, `3243` is the *position* in the DNA where the mutation occurs, and `tRNALeU(UUR)` is the *locus* associated to the mutation.

Two examples of clauses used for the annotation of the named entities *type* and *substitution* are the following:

[1] http://www.hmtdb.uniba.it/

```
substitution(X) ← follows(Y,X), type(Y).
type(X) ← distance(X,Y,3), position(Y),
  word_between(X,Y,‘‘nucleotide position’’).
```

Their interpretation is straightforward. The first clause states that a token `X` is labeled as *substitution* (i.e., which nucleotide is substituted by which other, `A` in `G` in the example text) if it is followed by a token `Y` which has been labeled as mutation *type* (`transition`). The second clause states that `X` is labeled as mutation *type* (`transition`) if it is three words far from a token `Y` that has been labeled as mutation *position* (`3243`) and there is an intermediate word `nucleotide position`.

It should be noted that in the above example, the first clause expresses a dependency between the annotation classes *type* and *substitution* of the same template of interest (mutation). As previously clarified, learning classification models which express these dependencies might lead to more accurate models, which reflect some co-occurrence of named entities in the text. Furthermore, when automated annotation is performed, context-sensitive recognition of named entities is possible thanks to learned models which reflect dependencies among annotation classes. However, discovering such concept dependencies poses additional problems for inductive learning. A brief description of an ILP learning algorithm that provides a solution to these problems is reported in Section 4.

3 Data Preparation

The dataset is composed by a set of manually annotated pre-categorized texts. Annotated texts are preprocessed by means of natural language facilities provided in the GATE (General Architecture for Text Engineering) system [6]. We exploit the ANNIE (A Nearly-New IE system) component which contains finite-state algorithms and the JAPE (a Java Annotation Patterns Engine) language which is also a finite-state transduction engine to recognize regular expressions. By means of ANNIE we perform tokenization, sentence splitting, part-of-speech tagging, general purpose named-entity recognition (e.g., persons, locations, organizations), and mapping into dictionaries.

We use both predefined dictionaries available with ANNIE (e.g., organization names, job title, geographical locations, dates, etc.) and domain-specific dictionaries that categorize biological entities such as diseases, enzymes, genes, and so on. General domain dictionaries are used to disambiguate some terms (e.g., places and geographical locations are useful to recognize terms about the ethnic origin of the diseased sample). Domain-specific dictionaries are flat dictionaries of canonical forms and variants of names that are peculiar of mitochondrial genetics. They include lists of names about diseases, genes, methods of analysis, nucleic acids, enzymes, and so on. They are exploited to reduce heterogeneity of data and to perform syntactic and semantic normalization such as a rough resolution of acronyms which in this domain are one of the sources of redundancy and ambiguity. For instance, recognizing that “myopathy, encephalopathy, lactic acidosis, and stroke-like episodes” and “mitochondrial encephalomyopathy lactic acidosis and strokelike episodes" are two variants of the same mitochondrial disease widely known by its acronym "MELAS" is possible when a disease dictionary is used.

JAPE grammars have been defined to identify appositions occurring in texts as well as some numeric and alphanumeric strings which are frequent in this domain. Finally, stopwords (e.g., articles, adverbs, and prepositions) are removed and stemming is performed by means of Porter's algorithm for English texts [22].

3.1 Data Representation

In this work the analysis units are sentences, which are, in their turn, composed of tokens. Each sentence or token is given a unique identifier (in the context of an abstract) based on its ordering within the given text. The relational representation of a sentence is described in terms of properties of occurring tokens and relations between them.

Properties, which are represented by unary descriptors, express statistical (e.g., token frequency), lexical (e.g., alphanumeric, capitalized token), structural (e.g., structure of complex tokens such as alphanumeric strings, abbreviations, acronyms, hyphenated tokens), syntactical (e.g., singular/plural proper/ not proper nouns, base/conjugated verbs) and domain-specific knowledge (e.g., an entity belonging to a dictionary). More precisely, the predicate `class` specifies the category of the described text (i.e., abstract, methods, results, etc.) and expresses information on the localization of annotations in documents. The predicate `word_to_string` maps an identifier to the corresponding stemmed token, `word_frequency` expresses the relative frequency of a token in the given text, `type_of` refers to morphological features and takes values in the set {`allcaps`, `mixedcaps`, `upperinitial`, `numeric`, `percentage`, `alphanumeric`, `real_number`}. Parts-of-speech are encoded by the predicate `type_pos`, while semantics is added by the `word_category` predicate.

Relations express structural properties such as the composition of sentences in passages of text and tokens in chunks or directly in sentences. Indeed, the following binary descriptors have been defined: `part_of`, which list tokens composing a sentence, and `follows`, which relates a token to its direct successor. Complex tokens (e.g., `A-*G`) are described by several predicates: the `s_part_of` relation on component tokens, the `first` and `last` predicates which define the first and second part of an hyphenated token respectively, the `length` predicate defining the length of component tokens, and some predicates (e.g., `middle_is_char`, `first_is_numeric`) defining the morphological nature of an alphanumeric string. Another form of relational knowledge concerns domain dictionaries and expresses the distance between two categorized tokens in the context of a sentence (`distance_word_category`).

In this work, we focus on the template *mutation*, which is composed of the following slots: *position* (i.e., position in the DNA where the mutation occurs), *type* (i.e., type of the mutation: insertion, deletion, translation, substitution, etc.), *type_position* (i.e., pieces of the DNA involved in the mutation and relative position in the DNA), *locus* (gene involved in the mutation), and *substitution* (i.e., type of substitution: which nucleotide is substituted by which other).

For the training data, only sentences containing at least a positive example of *position*, *type*, *type_position*, *locus* and *substitution* are considered. Henceforth, they are called *target sentences.* No relation between target sentences is currently considered, that is, the extraction of named entities remains local to sentences. An example of relational description generated for the target sentence reported in Section 2 is the following:[2]

```
annotation(3)=no_tag, ... annotation(7)=pathology,
annotation(8)=no_tag, ... annotation(13)=substitution,
annotation(14)=type, annotation(15)=no_tag, ...,
annotation(17)=position, annotation(18)=locus, ...,
annotation(30)=no_tag ←
class(2)=abstract, part_of(2,3)=true, ..., part_of(2,30)=true,
word_to_string(3)=cytoplast, ..., word_to_string(13)=a-*g,
s_part_of(13,31)=true, s_part_of(13,32)=true, first(13)=a,
last(13)=t, lenght(13)=4, single_char(31)=true,
single_char(32)=true, type_of(31)=allcaps, type_of(32)=allcaps,
word_to_string(14)=transition, ..., word_to_string(30)=cell,
type_of(3)=upperinitial, ..., type_of(29)=alphanumeric,
type_pos(3)=nnp, ..., type_pos(30)=nns, word_frequency(3)=1,
..., word_frequency(30)=2, word_category(7)=disease, ...,
word_category(9)=disease, ..., word_category(28)=nucleic_acid,
distance_word_category(7,9)=2, ..., distance_word_category(27,28)=1,
follows(3,4)=true, follows(4,5)=true, ..., follows(29,30)=true
```

The constant 2 denotes the sentence described in this clause, which belongs to an abstract of the collection, while the constants 3, 4, ..., 30 denote identifiers of tokens which are described in the body of the clause.

3.2 Background Knowledge

The background knowledge includes a transitive definition of the relation of "indirect successor":

```
tfollows(X,Y)=true ← follows(X,Y)=true
tfollows(X,Y)=true ← follows(X,Z)=true, tfollows(Z,Y)=true
```

as well as a number of clauses that express the synonymy between (stemmed) biological terms such as:

```
word_to_string(X)=transit ← word_to_string(X)=transversion
word_to_string(X)=substitut ← word_to_string(X)=replac
```

[2] Here, the first-order literals $p(X, Y)$ and $\neg p(X, Y)$ will be represented as $f_p(X,Y)=true$ and $f_p(X,Y)=false$, respectively, where f_p is the function symbol associated to the predicate p. This means that we deal with *classical negation*, ¬, but not with *negation by failure*, *not* [17].

The learning system used in this work makes the automated change of representation possible for training examples. This form of abstraction is very useful for tuning the representation of the training examples without acting on the procedures developed for text pre-processing. In this work, the following predicates are intensionally defined in the background knowledge:

```
char_number_char(X)=true ← first_is_char(X)=true,
    middle_is_numeric(X)=true, last_is_char(X)=true
number_char_char(X)=true ← first_is_numeric(X)=true,
    middle_is_char(X)=true, last_is_char(X)=true
char_char_number(X)=true ← first_is_char(X)=true,
    middle_is_char(X)=true, last_is_numeric(X)=true
```

They can appear in the body of learned clauses, while predicates on the morphological nature of alphanumeric strings (e.g., `first_is_char`, `middle_is_char`, etc.) cannot.

Finally, a typified form of both direct and transitive successor relations is also introduced. Some examples are reported in the following:

```
follows_string_jj(Y)=Z ←
    word_to_string(X)=Z, follows(X,Y)=true, type_pos(Y)=jj
follows_nn_string(X)=Z ←
    type_pos(X)=nn, follows(X,Y)=true, word_to_string(Y)=Z
tfollows_vb_nn(X,Y)=true ←
    type_pos(X)=vb, tfollows(X,Y)=true, type_pos(Y)=nn
tfollows_jj_nn(X,Y)=true ←
```

The first two clauses express the direct successor relations between a generic string and an adjective or a noun, while the last two clauses specify the transitive successor relations for verb-noun and adjective-noun pairs, respectively.

4 Learning Recursive Patterns

Tagging rules for automated entity extraction are automatically learned in the form of recursive logical theories which can naturally represent conceptual dependencies between named entities. Indeed multiple predicate learning (or multiple concept learning) and recursive theory learning are two faces of the same coin. In this application, each concept plays the role of an annotation class (i.e., template slot) and each textual object can be associated with at most one concept, i.e., concepts are considered mutually exclusive. The system used in this learning problem is ATRE[3] [18] which solves the following learning problem:

Given

- a set of concepts $K_1, K_2, \ldots, K_r$ to be learned,
- a set of observations O described in a language $\mathcal{L}_O$,
- a background knowledge BK described in a language $\mathcal{L}_{BK}$,

[3] http://www.di.uniba.it/~malerba/software/atre

- a language of hypotheses $\mathcal{L}_H$ that defines the space of hypotheses S_H
- a user's preference criterion *PC*,

Find
a (possibly recursive) logical theory $T \in S_H$, defining the concepts $K_1, K_2, \ldots, K_r$, such that T is complete and consistent with respect to the set of observations O and satisfies the preference criterion *PC*.

Both the language of hypotheses $\mathcal{L}_H$ and the language of background knowledge $\mathcal{L}_{BK}$ are limited to linked, range-restricted definite clauses [8]. Observations are represented as ground multiple-head clauses, called *objects*, which have a conjunction of literals in the head. Each object is associated with a unique object identifier (OID). The notion of multiple-head clauses in ATRE adapts the notion of *interpretation*, which is common to many relational data mining systems for efficiency reasons [9]. ATRE distinguishes objects from *examples*, which are described as pairs $\langle L,\ OID \rangle$, where L is a literal in the head of the object identified by *OID*. Examples can be considered *positive* or *negative*, according to the concept to be learned. For instance $\langle$*annotation(x10)=locus,* $O_1\rangle$ is a positive example of the concept `annotation(X)=locus` and a negative example of the concept `annotation(X)=type`. Actually, in this work, the set of concepts to be learned is defined by means of a set of literals of the type `annotation(X)=annotation_class`. No clause is generated for the concept `annotation(X)=no_tag`.

At the high-level ATRE implements a *sequential covering* algorithm [19]. A recursive theory T is built iteratively, starting from an empty theory T_0, and then adding a new clause at each iteration. In this way we obtain a sequence of theories:

$$T_0 = \emptyset, T_1, \ldots, T_i, T_{i+1}, \ldots, T_n = T$$

such that $T_{i+1} = T_i \cup \{C\}$ for some clause C and $LHM(T_i) \subseteq LHM(T_{i+1})$, where $LHM(T)$ denotes the least Herbrand model of a theory T. Let $pos(LHM(T))$ and $neg(LHM(T))$ be the number of positive and negative examples in $LHM(T)$, respectively. If we guarantee that:

1. $pos(LHM(T_i)) < pos(LHM(T_{i+1}))$, for each $i \in \{0, 1, \ldots, n-1\}$ and
2. $neg(LHM(T_i)) = 0$, for each $i \in \{0, 1, \ldots, n\}$,

then, after a finite number of iterations, a theory T, which is complete and consistent, is built. The first condition is guaranteed by selecting a positive example (or *seed*) $e^+ \notin LHM(T_i)$ of a concept K_j to be learned, and then by looking for a clause C, if any, such that $e^+ \in LHM(T_i \cup \{C\})$ (i.e., $pos(LHM(T_i \cup \{C\})) > pos(LHM(T_i))$). The second condition is more difficult to guarantee since the addition of a locally consistent clause C to a theory T_i does not preserve consistency of $T_i \cup \{C\}$ (non-monotonicity of the normal ILP setting). The approach followed in ATRE consists of simple syntactic changes in T_i, which eventually creates new *layers* [18].

The automated discovery of dependencies between concepts K_1, K_2, ..., K_r is based on a variant of the sequential covering learning strategy, which is traditionally adopted by single concept learning systems that generate clauses with the same literal in the head at each iteration. In multiple concept learning, clauses generated at each iteration may refer to different concepts. In addition, the body of the clause generated at the i-th iteration may involve any concept K_1, K_2, ..., K_r for which at least a clause has been added to the theory partially learned in previous iterations. In this way, dependencies between concepts can be generated.

At each iteration of the main loop of the sequential covering algorithm, clauses for distinct concepts are generated, and then one of them is picked. Since the generation of a clause depends on a seed, several seeds have to be chosen (if any, at least one seed per concept to be learned). Therefore, the search space is a forest of as many search-trees as the number of chosen seeds. Each search-tree is rooted with a unit clause and ordered by the generalization model adopted in ATRE (*generalized implication* [18]). The forest can be processed in parallel by as many concurrent tasks as the number of search-trees. Each task traverses the specialization hierarchy top-down through a sequential covering strategy, but synchronizes traversal with the other tasks at each level. Search proceeds toward deeper and deeper levels of the specialization hierarchies until at least a user-defined number of consistent clauses is found. Task synchronization is performed after that all "relevant" clauses at the same depth have been examined. A supervisor task decides whether the search should carry on or not on the basis of the results returned by the concurrent tasks. When the search is stopped, the supervisor selects the "best" consistent clause according to the user's preference criterion. This search strategy provides us with a solution to the problem of *interleaving* the induction of distinct concept definitions.

Actually, several special-purpose techniques have been designed specifically for the inductive synthesis of recursive logic theories. They are overviewed in [12]. As observed by Flener and Yilmaz, they are all non-incremental, i.e., the training examples are input all-at-once, therefore, the distinction of bottom-up versus top-down induction[4] does not really apply to them. Incremental techniques do not seem to be a promising research avenue for recursive theory learning (or, equivalently, for multiple concept learning), because they are often very sensitive to the ordering of the examples, which is not really adequate considering the fragile nature of recursive theories.

5 Experiments

We considered a data set of seventy-four papers concerning mithocondrial mutations selected for the annotation of HmtDB[5]. We considered the abstract of each paper and 228 target sentences out of 581 sentences. The total number of annotated tokens is 362, that is, 1.58 tokens per target sentence and 4.89 per

[4] In the bottom-up (top-down) approach the theory monotonically evolve from the maximally specific one (the maximally general one).

[5] The data set is available at http://www.di.uniba.it/∼malerba/software/atre/index.htm#Exp9

abstract. They correspond to about 9.3% of the total number of tokens that are described in the data set. The remaining tokens, that is 3004, are considered as *no tagged* tokens (i.e., as negative examples for all concepts to be learned).

The dataset is clearly imbalanced. However, it should be noted that the learning strategy implemented in ATRE is not affected by imbalanced data, since the goal is not to maximize the accuracy [23] but to generate consistent theories. Other systems that suffer from this problem may prove unsuitable for the task at hand, since they generate trivial classifiers.

Performances are evaluated by means of a 6-fold cross-validation, that is, the set of seventy-four abstracts is firstly divided into six folds (see Table 1), and then, for every fold, ATRE is trained on the remaining folds and tested on the hold-out fold. Results have been evaluated along several criteria. For each concept, we computed both the number of omission and commission errors and the value of precision and recall. *Omission* errors occur when annotations of tokens are missed, while *commission* errors occur when wrong annotations are "recommended" by some clause. The omission measure is reported as the ratio of the number of omission errors and the number of positive examples, while the commission measure as the ratio of the number of commission errors and the total number of examples. The *recall* measure is computed as the ratio of positive examples correctly annotated (i.e., true positives) and the sum of true positives and false negatives (i.e., omission errors). The *precision* measure is computed as the ratio of true positives and the sum of true positives and false positives (i.e., commission errors). Experimental results are reported in Table 2 for each fold. No omission error is reported for *type_position* when the third fold is held-out because there are no positive examples to test on.

Table 1. Distribution of examples per folds

Fold	**# sentences**	**# locus**	**# position**	**# substitution**	**# type**	**# type_position**	**# no_tag**	**# literals in body**
1	36	16	12	4	8	12	452	2424
2	40	27	13	2	5	4	546	2552
3	40	22	14	6	17	0	510	3098
4	34	16	6	5	17	23	517	3260
5	39	24	15	8	8	19	485	3083
6	27	14	6	2	6	31	494	3199
Total	228	119	66	27	61	89	3004	17793

The high variability among folds is mainly due to a heterogeneous distribution of examples that leads to different degrees of data sparseness. However, the percentage of commission errors is very low with respect to the percentage of omission errors (the system misses annotations rather than suggesting wrong annotations) independently of the fold. This means that learned rules are quite specific. By considering the complexity of learned theories (see Table 3), coverage rate can explain recall values. Best performances are obtained on the *substitution* class for which the system learns a more general and accurate

Table 2. Experimental results (percentage values): Average number and standard deviation of omission errors over positive ex., commission errors over negative ex., precision and recall

Fold	**locus**		**position**		**substitution**		**type**		**type_position**	
	omiss.	**comm.**	**omiss.**	**comm.**	**omiss.**	**comm.**	**omiss.**	**comm.**	**omiss.**	**comm.**
1	68.750	0.205	83.333	0.203	75	0	25	0.202	91.667	0.203
2	70.370	0.516	61.538	0.168	100	0	20	0.332	100	0
3	54.545	0.548	21.429	0	66.667	0	29.412	0.181	–	0.351
4	43.750	0.176	50	1.038	60	0.345	17.647	0	82.609	1.070
5	41.667	0.562	20	0.184	100	0.727	62.500	0.545	73.684	0.372
6	85.714	3.154	50	0	100	0	33.333	0.366	77.419	0.575
Avg	*60.799*	*0.860*	*47.717*	*0.266*	*83.611*	*0.179*	*31.315*	*0.271*	–	*0.428*
St.D.	*17.155*	*1.137*	*24.205*	*0.389*	*18.572*	*0.302*	*16.340*	*0.187*	–	*0.368*
	Avg	**St.D.**	**Avg**	**St.D.**	**Avg**	**St.D.**	**Avg**	**St.D.**	**Avg**	**St.D.**
Prec.	*77.305*	*18.387*	*81.355*	*25.258*	*87.778*	*19.052*	*65.972*	*23.632*	*74.009*	*37.073*
Rec.	*67.093*	*20.916*	*50.596*	*23.130*	*89.167*	*17.440*	*31.315*	*16.340*	–	–

Table 3. Complexity of the learned theories: number of positive examples over number of learned clauses per concept and average values

Fold	**locus**	**position**	**substitution**	**type**	**type_position**
1	16/35	12/23	4/3	8/34	12/15
2	27/30	13/23	2/2	5/2	4/17
3	22/34	14/20	6/1	17/30	0/16
4	16/32	6/19	5/2	17/26	23/13
5	24/27	15/17	8/2	8/27	19/14
6	14/36	6/24	2/2	6/22	31/11
Avg	**0.63**	**0.54**	**2.64**	**0.37**	**1.16**

theory. Indeed, examples of this class are the most homogeneous and the preprocessing module is able to produce discriminative descriptions. Conversely, worst performances of the system are related to the *type* class for which the lowest value of coverage rate is reported. Some low recall values and overfitted theories are due to the preprocessing module which is not completely apt to manage the variety of morpho-syntactic variations on the same term that affect this application domain. By scanning the learned theories, we observe that for some annotation classes, namely *locus* and *type_position*, many clauses do take into account only lexical information specified by the predicate *word_to_string*. Actually, learning tasks for these two classes appear to be intrinsically more complex since we observe the highest percentage of commission errors despite the the highest percentage of positive examples available. As regards the percentage of omission errors, we notice that while it is positively correlated to the number of discovered clauses, it results uncorrelated to the number of positive examples.

We adopt the same experimental setting to run the system by disabling the search for recursive definitions in the space of clauses. In this experiment, the system explores a specialization hierarchy for one concept at a time and learned theories are independently generated. Results are reported in Table 4. By comparing precision and recall values observed in the two experiments, we conclude that recursive theory learning improves performances for both *locus* and *type* classes, while it does not affect the results for the *position* and the *substitution* classes. This observation justifies the computational effort spent for learning recursive theories in this IE task.

Table 4. Experimental results obtained by disabling recursion (percentage values): Average number and standard deviation of omission errors over positive ex., commission errors over negative ex., precision and recall

Fold	**locus**		**position**		**substitution**		**type**		**type_position**	
	omiss.	**comm.**	**omiss.**	**comm.**	**omiss.**	**comm.**	**omiss.**	**comm.**	**omiss.**	**comm.**
1	75	0.205	83.33	0.203	75	0	75	0.202	100	0.203
2	59.26	0.516	61.54	0.168	100	0	80	0.332	100	0
3	63.64	0.731	21.43	0	66.67	0	35.29	0.181	–	0.351
4	62.50	0.880	50.00	1.038	60	0.345	29.41	0	91.30	0.891
5	41.67	0.562	20	0.184	100	0.727	62.50	0.545	63.16	0
6	64.29	2.597	50.00	0	100	0	50	0.366	64.52	0.575
Avg	*61.058*	*0.915*	*47.717*	*0.266*	*83.611*	*0.179*	*55.368*	*0.271*	–	*0.337*
St.D.	*20.729*	*0.187*	*24.205*	*0.389*	*18.572*	*0.302*	*10.888*	*0.855*	–	*0.349*
	Avg	**St.D.**	**Avg**	**St.D.**	**Avg**	**St.D.**	**Avg**	**St.D.**	**Avg**	**St.D.**
Prec.	*72.836*	*18.568*	*81.355*	*25.258*	*87.778*	*19.052*	*76.766*	*16.041*	*76.672*	*38.299*
Rec.	*61.058*	*10.888*	*50.596*	*23.130*	*89.167*	*17.440*	*55.368*	*20.729*	–	–

For the sake of completeness, some clauses learned by ATRE have been analyzed. Some of them follow:

```
annotation(X1)=type_position ← char_number_char(X1)=true
annotation(X1)=type_position ← tfollows_string_nn(X2)=trnaser,
  type_of(X1)=alphanumeric
annotation(X1)=position ← follows(X2,X1)=true,
  type_of(X1)=numeric, follows(X1,X3)=true,
  word_category(X3)=gene, word_to_string(X2)=position
```

The first clause states that `X1` is labeled as *type_position* if it is an alphanumeric token composed by a char, a number and another char. This is one of the first clauses that ATRE adds to the learned theory and covers many examples. Actually, information on *type_position* of a mutation are tokens such as $A1262G$, that means that A is substituted by G at position 1262 of the DNA. The second clause concerns the same concept and it states that `X1` is labeled as *type_position* if it is an alphanumeric token which is followed by the string `trnaser`. This matches patterns where *type_position* information occurs in the neighborhood of gene names (e.g., `trnaser`). The third clause states that `X1` is labeled as *position* if it is a numeric token which succeeds the token "position" and precedes a token of the “gene” category. This clause captures patterns like “an A-to-G mutation at *position 3426 (tRNALeu)*”.

Meaningful dependencies have been also discovered such as the following:

```
annotation(X1)=type ← follows(X1,X2)=true,
  word_frequency(X2) ∈ [8..140],
  follows(X3,X1)=true, annotation(X3)=substitution
```

It states that $X1$ is annotated as *type* if it precedes a frequent token and succeeds another token which has been annotated as *substitution*. Another example of discovered concept dependency is the following:

```
annotation(X1)=position ← follows(X2,X1)=true
  annotation(X2)=substitution, follows(X3,X1)=true,
  follows(X1,X4)=true, word_frequency(X4) ∈ [6..6],
  annotation(X3)=type, follows(X1,X5)=true,
  annotation(X5)=locus, word_frequency(X1) ∈ [1..2]
```

This clause states that $X1$ is annotated as *position* if it succeeds two tokens which have been annotated as *type* and *substitution*, respectively. Moreover it precedes a token occurring about 6 times in the abstract and that is followed by a *locus* annotation. Finally, $X1$ is quite infrequent in the abstract. It matches text portions like the following: “a G-to-A (X2) transition (X3) at nucleotide pair 14459 (X1), changed a moderately conserved alanine to a valine at NADH (X4) dehydrogenase subunit 6 (ND6) (X6) residue 72”.

6 Conclusions

ILP provides appropriate computational solutions for problems posed by biomedical IE thanks to its adequacy to work with first-order logic representations of texts and to its suitability to take advantage of the abundant domain knowledge. *Template filling* tasks appear to be especially challenging for ILP, since they raise problems that are peculiar of link-learning tasks, where examples, in addition to their inherent relational structure, present relations to other examples [14]. Intuitively, template filling operations can be simplified when tagging of named entities is performed by considering some conceptual dependencies implicitly defined among entities of the same template. In this paper, we have proposed an application of recursive theory learning to a real-world IE task on the biomedical literature. Recursive patterns are discovered by inducing mutually dependent definitions of concepts by means of the ILP system ATRE. This system allows us to discover meaningful patterns among biomedical entities of interest. Results obtained on a limited number of documents show that high performances are obtained when descriptions of examples are safe from inconsistencies due to morpho-syntactic variations that are not completely handled by the preprocessing module. Moreover, results show that when learned theories can express mutual dependencies between concepts tagging performances improves.

As future work, further experiments on some recently made available biomedical datasets for ILP [7] will be conducted. We also plan to examine benefits of discovering template slot dependencies in reconstructing records of a complete template filling task. Finally, the application of a transductive framework [25] is worth to be investigated because of the high disproportion between the number of labeled documents and that of unlabeled documents available in IE tasks.

Acknowledgments

This work has been funded by the IBM Faculty Award 2004 received from IBM Corporation to promote collaborative research on "Knowledge Discovery Technologies for the development of a Gene and SNPs filtering engine". The authors would like to thank Prof. Marcella Attimonelli for her assistance in the use of HmtDB and Antonio Varlaro for his support in the use of the system ATRE.

References

1. Aitken, J.S.: Learning information extraction rules: An inductive logic programming approach. In: Proceedings of the 15th European Conference on Artificial Intelligence, pp. 355–359 (2002)
2. Attimonelli, M., Accetturo, M., Santamaria, M., Lascaro, D., Scioscia, G., Pappada, G., Tommaseo-Ponzetta, M., Torroni, A.: Hmtdb, a human mitochondrial genomic resource based on variability studies supporting population genetics and biomedical research. BMC Bioinformatics 1(6) (2005)
3. Berardi, M., Ceci, M., Malerba, D.: A hybrid strategy for knowledge extraction from biomedical documents. In: ICDAR workshop on "Neural Networks and Learning in Document Analysis and Recognition", Seoul, Korea (2005)
4. Califf, M.E., Mooney, R.J.: Relational learning of pattern-match rules for information extraction. In: AAAI '99/IAAI '99, pp. 328–334. American Association for Artificial Intelligence (1999)
5. Craven, M., Kumlien, J.: Constructing biological knowledge bases by extracting information from text sources. In: Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology, pp. 77–86. AAAI Press, Stanford (1999)
6. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: Gate: A framework and graphical development environment for robust nlp tools and application. In: Proc. of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02), Philadelphia, USA (2002)
7. Cussens, J., Nedellec, C(ed.): Proceedings of the 4th ICML Workshop on Learning Language in Logic (LLL05), Bonn, Germany (2005)
8. Cussens, J., Nedellec, C. (ed.): In: Proceedings of the 4th ICML Workshop on Learning Language in Logic (LLL05), Bonn, Germany (2005)
9. Džeroski, S., Lavrač, N.: Relational Data Mining. Springer-Verlag, Heidelberg (2001)
10. Dzeroski, S., Cussens, J., Manandhar, S.: An introduction to inductive logic programming and learning language in logic. In: Cussens, J., Džeroski, S. (eds.) Learning Language in Logic. LNCS (LNAI), vol. 1925, pp. 3–35. Springer, Heidelberg (2000)

11. Ferilli, S., Fanizzi, N., Semeraro, G.: Learning logic models for automated text categorization. In: AI*IA 01. Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence, London, UK, pp. 81–86. Springer, Heidelberg (2001)
12. Flener, P., Yilmaz, S.: Inductive synthesis of recursive logic programs: achievements and prospects. Journal of Logic Programming, Special Issue on Synthesis, Transformation, and Analysis 41(2-3), 141–195 (1999)
13. Freitag, D.: Toward general-purpose learning for information extraction. In: Proceedings. of the 17th int. conf. on Computational linguistics, pp. 404–408, Morristown, NJ, USA, Association for Computational Linguistics (1998)
14. Goadrich, M., Oliphant, L., Shavlik, J.W.: Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In: Proceedings of the Fourteenth International Conference on Inductive Logic Programming, pp. 98–115 (2004)
15. Hirschman, L., Yeh, A., Blaschke, C., Valencia, A.: Overview of biocreative: critical assessment of information extraction for biology. Bioinformatics 6 (2005)
16. Junker, M., Sintek, M., Rink, M.: Learning for text categorization and information extraction with ilp. In: Learning Language in Logic, pp. 247–258 (1999)
17. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer-Verlag, Heidelberg (1987)
18. Malerba, D.: Learning recursive theories in the normal ilp setting. Fundamenta Informaticae 57(1), 39–77 (2003)
19. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
20. Mooney, R.: Learning for semantic interpretation: Scaling up without dumbing down. In: Cussens, J. (ed.) Proc. of the 1st Workshop on Learning Language in Logic, Bled, Slovenia, pp. 7–15 (1999), `citeseer.ist.psu.edu/mooney99learning.html`
21. Loveland, D.W. (ed.): Machine Learning for Information Extraction in Genomics - State of the art and perspectives. LNCS, vol. 138. Springer, Heidelberg (1982)
22. Porter, M.F.: Readings in information retrieval, chapter An algorithm for suffix stripping, pp. 313–316 (1997)
23. Provost, F.: Learning with imbalanced data sets (invited paper). In: Proc. of AAAI'2000 Workshop on Imbalanced Data Sets (2000)
24. Shatkay, H., Feldman, R.: Mining the biomedical literature in the genomic era: an overview. Journal of Computational Biology 10, 821–855 (2003)
25. Vapnik, V.: Statistical Learning Theory. Wiley, New York (1998)