Markov Logic Networks for Document Layout Correction

Stefano Ferilli, Teresa M.A. Basile, and Nicola Di Mauro

Department of Computer Science, LACAM laboratory
University of Bari "Aldo Moro", via Orabona 4, 70125 - Bari,
{ferilli, basile, ndm}@di.uniba.it

Abstract. The huge amount of documents in digital formats raised the need of effective content-based retrieval techniques. Since manual indexing is infeasible and subjective, automatic techniques are the obvious solution. In particular, the ability of properly identifying and understanding a document's structure is crucial, in order to focus on the most significant components only. Thus, the quality of the layout analysis outcome biases the next understanding steps. Unfortunately, due to the variety of document styles and formats, the automatically found structure often needs to be manually adjusted. In this work we present a tool based on Markov Logic Networks to infer corrections rules to be applied to forthcoming documents. The proposed tool, embedded in a prototypical version of the document processing system DOMINUS, revealed good performance in real-world experiments.

1 Introduction

The task aimed at identifying the geometrical structure of a document is known as Layout Analysis, and represents a wide area of research in document processing, for which several solutions have been proposed in literature. The quality of the layout analysis outcome is crucial, because it determines and biases the quality of the next understanding steps. Unfortunately, the variety of document styles and formats to be processed makes the layout analysis task a non-trivial one, so that the automatically found structure often needs to be manually fixed by domain experts.

The geometric layout analysis phase involves several processes, among which page decomposition. Several works concerning the page decomposition step are present in the literature, exploiting different approaches and having different objectives. Basic operators of all these approaches are split and merge: they exploit the features extracted from an elementary block to decide whether splitting or merging two or more of the identified basic blocks in a top-down [8], bottom-up [15] or hybrid approach [12] to the page decomposition step. Since all methods split or merge blocks/components based on certain parameters, parameter estimation is crucial in layout analysis. All these methods exploit parameters that are able to model the split or merge operations in specific classes of the document domain. Few adaptive methods, in the sense that split or merge operations are performed using estimated parameter values, are present in the literature [2, 10]. A step forward is represented by the exploitation of Machine Learning techniques in order to automatically assess the parameters/rules able to perform the document page decomposition, and hence the eventual correction of the performed split/merge operations, without requiring an empirical evaluation on the specific document domain at hand. To this regard, learning methods have been used to separate textual areas from graphical areas [5] and to classify text regions as headline, main text, etc. [3, 9] or even to learn split/merge rules in order to carry out the corresponding operations and/or correction [11, 16].

However, a common limit of the above reported methods regards the consideration that they are all designed with the aim of working on scanned documents, and in some cases on documents of a specified typology, thus lacking any generality of the proposal with respect to the online available documents that can be of different digital formats. On the other hand, methods that work on natively digital documents assume that the segmentation phase can be carried out by simply performing a matching of the document itself with a standard template, even in this case, of a specified format. In this work we propose the application of a Statistical Relational Learning [7] (SRL) technique to infer a probabilistic logical model recognising wrong document layouts from sample corrections performed by expert users in order to automatically apply them to future incoming documents. Corrections are codified in a first-order language and the learned correction model is represented as a Markov Logic Network [14] (MLN). Experiments in a real-world task confirmed the good performance of the solution.

2 Preliminaries

In this section we briefly describe DOC (Document Organization Composer) [4], a tool for discovering a full layout hierarchy in digital documents based primarily on layout information. The layout analysis process starts with a preprocessing step performed by a module that takes as input a generic digital document and extracts the set of its elementary layout components (*basic-blocks*), that will be exploited to identify increasingly complex aggregations of basic components.

The first step in the document layout analysis concerns the identification of rules to automatically shift from the basic digital document description to a higher level one. Indeed, the basic-blocks often correspond just to fragments of words (e.g., in PS/PDF documents), thus a preliminary aggregation based on their overlapping or adjacency is needed in order to obtain blocks surrounding whole words (*word-blocks*). Successively, a further aggregation of word-blocks could be performed to identify text lines (*line-blocks*). As to the grouping of blocks into lines, since techniques based on the mean distance between blocks proved unable to correctly handle cases of multi-column documents, Machine Learning approaches were applied in order to automatically infer rewriting rules that could suggest how to set some parameters in order to group together rectangles (words) to obtain lines. To do this, a kernel-based method was exploited to learn rewriting rules able to perform the bottom-up construction of the whole document starting from the basic/word blocks up to the lines.

The next step towards the discovery of the high-level layout structure of a document page consists in applying an improvement of the algorithm reported in [1]. To this aim, DOC analyzes the whitespace and background structure of each page in the document in terms of rectangular covers and identifies the white rectangles that are present in the page by decreasing area, thus reducing to the Maximal White Rectangle problem as follows: given a set of rectangular content blocks (*obstacles*) $C = \{r_0, \ldots, r_n\}$, all placed inside the page rectangular contour r_b, find a rectangle r contained in r_b whose area is maximal and that does not overlap any $r_i \in C$. The algorithm exploits a priority queue of pairs (r, O), where r is a rectangle and O is a set of obstacles overlapping r. The priority of the pair in the queue corresponds to the area of its rectangle. Pairs are iteratively extracted from the queue and if the set of obstacles corresponding to its rectangle is empty, then it represents the maximum white rectangle still to be discovered. Otherwise, one of its obstacles is chosen as a *pivot* and the rectangle is consequently split into four regions (above, below, to the right and to the left of the pivot). Each such region, along with the obstacles that fall in it, represents a new pair to be inserted in the queue. Complement of the found maximal white rectangles yield the document content blocks.

However, taking the algorithm to its natural end and then computing the complement would result again in the original basic blocks, while the layout analysis process aims at returning higher-level layout aggregates. This raised the problem of identifying a stop criterion to end this process. An empirical study carried out on a set of 100 documents of three different categories revealed that the best moment to stop the algorithm is when the ratio of the last white area retrieved with respect to the total white area in the current page of the document decreases up to 0, since before it the layout is not sufficiently detailed, while after it useless white spaces are found.

3 Learning Layout Correction Theories

The proposed strategy for automatically assessing a threshold to decide when stopping the background retrieval loop allows to immediately reach a layout that is already good for many further tasks of document image understanding. Nevertheless, such a threshold is clearly a trade off between several document types and shapes, and hence in some cases the layout needs to be slightly improved through a fine-tuning step that must be specific for each single document. To handle this problem, a tool was provided that allows the user to directly point out useful background fragments that were not yet retrieved and add them explicitly (*white forcing*), or, conversely, to select useless ones that were erroneously retrieved and remove them from the final layout (*black forcing*).

The forcing functionality allows the user to interact with the layout analysis algorithm and suggest which specific blocks are to be considered as background or content. To see how it can be obtained, let us recall that the algorithm, at each step, extracts from the queue a new area to be examined and can take three actions correspondingly: if the contour is not empty, it is split and the resulting fragments are enqueued; if the contour is empty and fulfils the constraints, it is added to the list of white areas; if the contour is empty but does not fulfil the constraints, it is discarded.

Allowing the user to interact with the algorithm means modifying the algorithm behaviour as a consequence of his choices. It turns out that the relevance of a (white or black) block to the overall layout can be assessed based on its position inside the document page and its relationships with the other layout components. According to this assumption, each time the user applies a manual correction, the information on his actions and on their effect can be stored in a trace file for subsequent analysis. In particular, each manual correction (user intervention) can be exploited as an example from which learning a model on how to classify blocks as meaningful or meaningless for the overall layout. Applying the learned models in subsequent incoming documents, it would be possible to automatically decide whether or not any white (resp., black) block is to be included as background (resp., content) in the final layout, this way reducing the need for user intervention.

3.1 Description Language

Now let us turn to the way in which the trace of manual corrections are codified. The assumption is that the user changes the document layout when he considers that the proposed layout is wrong, then it forces a specific block because he knows the resulting effect on the document and considers it as satisfactory. Thus, to properly learn rules that can help in automatically fixing and improving the document layout analysis outcome, one must consider what is available before the correction takes place, and what will be obtained after it is carried out. For this reason, each example, representing a correction, will include a description of the blocks' layout both before and after the correction. However, the modification is typically *local*, i.e. it does not affect the whole document layout, but involves just a limited area surrounding the forced block. This allows to limit the description to just such an area. To sum up, the log file of the manual corrections, applied by the user after the execution of the layout analysis algorithm, will include both the white and the black blocks he forced, and will record, for each correction, information about the blocks and frames surrounding the forced block, both before and after the correction.

In the following `b` stands for block, `f` for frame and `r` for rectangle. Each log example is represented as a set of first-order predicates and it is labelled as positive for forcing black (`merge(b)`) or white block (`split(b)`). Negative examples are denoted by negating the predicate with the `not` predicate.

A set of facts describing the other blocks in the area of interest before and after the correction are reported. It contains information on the document page in which the correction took place, the horizontal/vertical size and position of a block in the overall document, whether it is at the beginning, in the middle or at the end of the document, furthermore the forced block and the layout situation both before and after the correction are represented in the log example. The description of each of the two situations (before and after the correction) is based on literals expressing the page layout and describing the blocks and frames surrounding the forced block, and, among them, only those touching or overlapping the forced block. Each involved frame `frame(r)` or block `block(r)` is considered as a rectangular area of the page, and described according to the following parameters: horizontal and vertical position of its centroid with respect to the top-left corner of the page (`posX(r,x)` and `posY(r,y)`), height and width (`width(r)` and `height(r)`), and its content type (`type(r,t)`).

The relationships between blocks/frames are described by means of a set of predicates representing the spatial relationships existing among all considered frames and among all blocks belonging to the same frame (`belongs(b,f)`), that touch or overlap the forced block; furthermore for each frame or block that touches the forced block a literal specifying that they touch (`touches(b1,b2)`); finally, for each block of a frame that overlaps the forced block the percentage of overlapping (`overlaps(b1,b2,perc`). It is fundamental to completely describe the mutual spatial relationships among all involved elements. All, and only, the relationships between each block/frame and the forced blocks are expressed, but not their inverses (i.e., the relationships between the forced block and the block/frame in question). To this aim, the model proposed in [13] for representing the spatial relationships among the blocks/frames was considered. Specifically, according to such a model, fixed a rectangle, its plane is partitioned in 25 parts and its spatial relationships to any other rectangle in the plane can be specified by simply listing the parts to which the other rectangle overlaps (`overlapPart(r1,r2,part)`).

3.2 Markov Logic Networks background

Here, we briefly introduce the notions of a SRL approach combining first-order logic and probabilistic graphical models in a single representation. SRL seeks to combine the power of statistical learning to deal with the presence of uncertainty in many real-world application domains, and the power of relational learning in dealing the complex structure of such domains. We provide the background on Markov Logic Networks [14] (MLNs) representing first-order knowledge base with a weight attached to each formula.

A Markov Network (MN) models the joint distribution of a set of variables $X = (X_1, \ldots, X_n)$ made up of an undirected graph G and a set of potential functions ϕ_k. Each variable is represented with a node in the graph, and the model has a potential function for each clique in the graph. The joint distribution represented by a MN is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}),$$

where $x_{\{k\}}$ is the state of the k-th clique, and the partition function Z is given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. MNs may be represented as log-linear models, where each potential clique is replaced by an exponentiated weighted sum of features of the state, as in the following formula

$$P(X = x) = \frac{1}{Z} \exp \left(\sum_j w_j f_j(x) \right).$$

A first-order knowledge base (KB) is a set of formulas in first-order logic constructed using constants, variables, functions and predicates. A formula is satisfiable iff there exists at least one world (Herbrand interpretation) in which it is true. A first-order KB can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability, and MLNs soften these constraints. When a world violates one formula it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. In MLNs each formula has an associated weight representing how strong a constraint is. An high weight corresponds to a great difference in log probability between a world that satisfies the formula and one that does not.

More formally, A MLN L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, \ldots, c_n\}$, it defines a MN $M_{L,C}$ that a) contains one binary node[1] for each possible grounding of each predicate appearing in L, and one feature[2] for each possible grounding of each formula F_i in L, whose weight is the w_i associated with F_i in L. A MLN can be viewed as a template for constructing MNs, named ground Markov networks. The probability distribution over possible worlds x specified by the ground MN $M_{L,C}$ is given by

$$P(X = x) = \frac{1}{Z} \exp \left(\sum_i w_i n_i(s) \right) = \frac{1}{Z} \prod_i \phi_i(x_{\{i\}})^{n_i(x)}, \qquad (1)$$

where $n_i(x)$ is the number of true groundings of F_i in x, $x_{\{i\}}$ is the state (truth values) of the atoms appearing in F_i, and $\phi_i(x_{\{i\}}) = e^{w_i}$.

Reasoning with MLNs can be classified as either learning or inference. Inference in SRL is the problem of computing probabilities to answer specific queries after having defined a probability distribution. Learning corresponds to infer both the structure and the parameters of the true unknown model. An inference task is computing the probability that a formula holds, given an MLN and set of constants, that, by definition, is the sum of the probabilities of the worlds where it holds. MLN weights can be learned generatively by maximizing the likelihood of a relational database consisting of one or more *possible worlds* that form our training examples. The inference and learning algorithms for MLNs are publicly available in the open-source Alchemy system[3]. Given a relational database and a set of clauses in the KB, many weights learning and inference procedures are implemented in the Alchemy system. For weight learning, we used

[1] The value of the node is 1 if the ground atom is true, and 0 otherwise.
[2] The value of the feature is 1 if the ground formula is true, and 0 otherwise.
[3] http://alchemy.cs.washington.edu/

the generative approach tha maximise the pseudo-likelihood of the data with standard Alchemy parameters (`./learnwts -g`), while for inference, we used the MaxWalkSAT procedure (`./infer`) with standard Alchemy parameters.

Predicates `split(b)` and `merge(b)` represent the query in our MLN, where `b` is the forced block. The goal is to assign a black (merge) or white (split) forcing to unlabelled blocks. The MLN clauses used in our system are reported in the following. There is one of the following rule for each of the 25 planes capturing the spatial relationships among the blocks:

```
overlapPart(b1,b,part) => split(b), merge(b)
```

Other relations are represented by the MLN rules:

```
belongs(b1,b) => split(b), merge(b)
touches(b1,b) => split(b), merge(b)
overlaps(b1,b,perc) => split(b), merge(b)
belongs(b1,b) => split(b1), merge(b1)
touches(b1,b) => split(b1), merge(b1)
overlaps(b1,b,perc) => split(b1), merge(b1)
```

Running weight learning with Alchemy, we will learn a weight for every clause representing how good a relation is for predicting the label. Then, whit this classifier, each test instance can be classified using inference.

4 Experimental evaluation

The proposed description language was used to run an experiment aimed at checking whether it is possible to learn a theory that can profitably automatize, at least partially, the layout correction process. Two target concepts were considered: *split* (corresponding to the fact that a block discarded or not yet retrieved by the layout analysis algorithm must be forced to belong to the background) and *merge* (corresponding to the fact that a white rectangle found by the layout analysis algorithm must, instead, be discarded). A 10-fold cross-validation technique was exploited to obtain the training and test sets.

The experimental dataset concerned the corrections applied to obtain the correct layout on about one hundred documents, evenly distributed in four categories. According to the strategy described above, the examples concern significant background blocks that were not retrieved (split) or useless white blocks erroneously considered as background (merge) by the basic layout analysis algorithm. For the first dataset this activity resulted in a set of 786 examples of block correction, specifically 263 for split and 523 for merge. Positive examples for split were considered as negative for merge and *vice versa*, this way exploiting the whole dataset. Thus, each single correction was interpreted from two perspectives: as a positive example for the kind of forcing actually carried out by the user, and additionally as a negative example for the other kind of forcing.

The performances of the algorithm is evaluated by computing the area under the Receiver Operating Characteristic (ROC) curve, which shows how the number of correctly classified positive examples varies with the number of incorrectly classified negative examples, and the Precision-Recall (PR) curve.

Table 1. Area under the ROC and PR curves for split (S) and merge (M).

		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	Mean ± St.Dev.
S	ROC	0.992	0.961	0.913	0.910	0.964	0.917	0.916	0.984	0.968	0.940	0.946 ± 0.031
	PR	0.989	0.968	0.810	0.868	0.949	0.916	0.762	0.967	0.946	0.852	0.903 ± 0.076
M	ROC	0.941	0.971	0.964	0.950	0.966	0.950	0.978	0.952	0.952	0.920	0.954 ± 0.017
	PR	0.966	0.983	0.933	0.970	0.985	0.972	0.989	0.934	0.963	0.930	0.962 ± 0.023

Table 1 reports the results for the queries split and merge in this first experiment. The outcomes of the experiment suggest that the description language proposed and the way in which the forcings are described are effective to let the system learn clause weights that can be successfully used for automatic layout correction. This suggested to try another experiment to simulate the actual behavior of such an automatic system, working on the basic layout analysis algorithm. After finishing the execution of the layout analysis algorithm according to the required stop threshold, three queues are produced (the queued areas still to be processed, the white areas discarded because not satisfying the constraints and the white blocks selected as useful background). Among these, the last one contains whites that can be forced to black, while the other two contain rectangles that might be forced white.

Since the rules needed by DOC to automatize the layout correction process must be able to evaluate each block in order to decide whether forcing it or not, it is not sufficient any more to consider each white block forcing as a counterexample for black forcing and *vice versa*, but to ensure that the learned MLN is correct, also all blocks in the document that have not been forced must be exploited as negative examples for the corresponding concepts. The adopted solution was to still express forcings as discussed above, including additional negative examples obtained from the layout configuration finally accepted by the user. Indeed, when the layout is considered correct, all actual white blocks that were not forced become negative examples for concept merge (because they could be forced as black, but weren't), while all white blocks, discarded or still to be processed become negative examples for the concept split (because they weren't forced). The dataset for this experiment was obtained by running the layout analysis algorithm until the predefined threshold was reached, and then applying the necessary corrections to fix the final layout. The 36 documents considered were a subset of the former dataset, evenly distributed among the four categories. Specifically, the new dataset included 113 positive and 840 negative examples for merge, and resulted in the performance reported in Table 2.

As to the concept split, the dataset was made up of 101 positive and 10046 negative examples. The large number of negative examples is due to the number of white blocks discarded or still to be processed being typically much greater

Table 2. Area under the ROC and PR curves for split (S) and merge (M).

		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	Mean ± St.Dev.
S	ROC	0.920	0.987	0.959	0.969	0.977	0.925	0.989	0.982	0.915	0.977	0.960 ± 0.029
	PR	0.841	0.917	0.699	0.909	0.692	0.914	0.778	0.774	0.807	0.913	0.824 ± 0.088
M	ROC	1.000	0.998	1.000	0.964	1.000	0.973	1.000	0.996	0.994	0.996	0.992 ± 0.013
	PR	1.000	0.986	1.000	0.804	1.000	0.874	1.000	0.971	0.966	0.967	0.957 ± 0.066

than that of white blocks found. Since exploiting such a large number of negative examples might have significantly unbalanced the learning process, only a random subset of 843 such examples was selected, in order to keep the same ratio between positive and negative examples as for the merge concept. The experiment run on such a subset provided the results shown in Table 2.

Figure 1 reports the plot obtained averaging ROC and PR curves for the ten folds. As reported in [6], a technique to evaluate a classifier over the results obtained with a cross validation method is to merge together the test instances belonging to each fold with their assigned scores into one large test set.

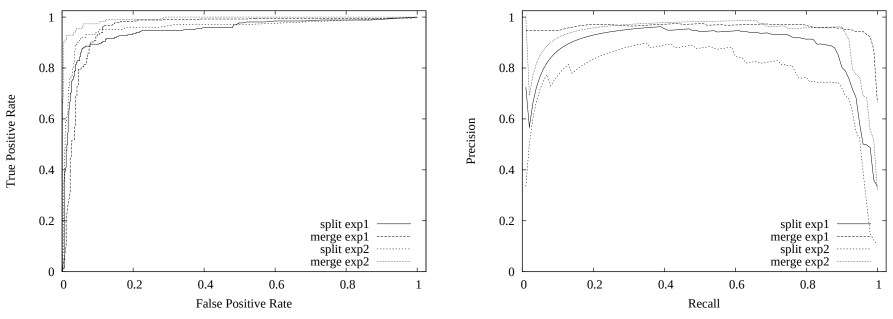

Fig. 1. ROC (left) and PR (right) curve by merging the 10-fold curves.

5 Conclusions

The variety of document styles and formats to be processed makes the layout analysis task a non-trivial one and often the automatically found structure often needs to be manually fixed by domain experts. In this work we proposed a tool able to use the steps carried out by the domain expert, with the aim of correcting the outcome of the layout analysis phase, in order to infer models to be applied to future incoming documents. Specifically, the tool makes use of a first-order logic representation of the document structure as it is not fixed and a correction often depends on the relationships of the wrong components with the surrounding ones. Moreover, the tool exploits the statistical relational learning system Alchemy.

Experiments in a real-world domain made up of scientific documents have been presented and discussed, showing the validity of the proposed approach.

References

1. Breuel, T.M.: Two geometric algorithms for layout analysis. In: Proceedings of the 5th International Workshop on Document Analysis Systems (DAS). Lecture Notes in Computer Science, vol. 2423, pp. 188–199. Springer (2002)
2. Chang, F., Chu, S.Y., Chen, C.Y.: Chinese document layout analysis using adaptive regrouping strategy. Pattern Recognition 38(2), 261–271 (2005)
3. Dengel, A., Dubiel, F.: Computer understanding of document structure. International Journal of Imaging Systems and Technology 7, 271–278 (1996)
4. Esposito, F., Ferilli, S., Basile, T.M.A., Di Mauro, N.: Machine Learning for digital document processing: from layout analysis to metadata extraction. In: Marinai, S., Fujisawa, H. (eds.) Machine Learning in Document Analysis and Recognition, Studies in Computational Intelligence, vol. 90, pp. 105–138. Springer (2008)
5. Etemad, K., Doermann, D., Chellappa, R.: Multiscale segmentation of unstructured document pages using soft decision integration. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(1), 92–96 (1997)
6. Fawcett, T.: Roc graphs: Notes and practical considerations for researchers. Tech. rep., HP Laboratories (2004), http://www.hpl.hp.com/techreports/2003/HPL-2003-4.pdf
7. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning). The MIT Press (2007)
8. Krishnamoorthy, M., Nagy, G., Seth, S., Viswanathan, M.: Syntactic segmentation and labeling of digitized pages from technical journals. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(7), 737–747 (1993)
9. Laven, K., Leishman, S., Roweis, S.: A statistical learning approach to document image analysis. In: Proceedings of the Eighth International Conference on Document Analysis and Recognition. pp. 357–361. IEEE Computer Society (2005)
10. Liu, J., Tang, Y.Y., Suen, C.Y.: Chinese document layout analysis based on adaptive split-and-merge and qualitative spatial reasoning. Pattern Recognition 30(8), 1265–1278 (1997)
11. Malerba, D., Esposito, F., Altamura, O., Ceci, M., Berardi, M.: Correcting the document layout: A machine learning approach. In: Proceedings of the 7th Intern. Conf. on Document Analysis and Recognition. pp. 97–103. IEEE Comp. Soc. (2003)
12. Okamoto, M., Takahashi, M.: A hybrid page segmentation method. In: Proceedings of the Second International Conference on Document Analysis and Recognition. pp. 743–748. IEEE Computer Society (1993)
13. Papadias, D., Theodoridis, Y.: Spatial relations, minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science 11(2), 111–138 (1997)
14. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)
15. Simon, A., Pret, J.C., Johnson, A.P.: A fast algorithm for bottom-up document layout analysis. IEEE Transactions on PAMI 19(3), 273–277 (1997)
16. Wu, C.C., Chou, C.H., Chang, F.: A machine-learning approach for analyzing document layout structures with two reading orders. Pattern Recogn. 41(10), 3200–3213 (2008)