Automatic Document Layout Analysis through Relational Machine Learning

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

Abstract. The current spread of digital documents 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. At a geometrical level, this task is known as Layout Analysis, and thoroughly studied in the literature. On suitable descriptions of the document layout, Machine Learning techniques can be applied to automatically infer models of classes of documents and of their components. Indeed, organizing the documents on the grounds of the knowledge they contain is fundamental for being able to correctly access them according to the user's needs.

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. We propose the application of supervised Machine Learning techniques to infer correction rules to be applied to forthcoming documents. A first-order logic representation is suggested, because corrections often depend on the relationships of the wrong components with the surrounding ones. Moreover, as a consequence of the continuous flow of documents, the learned models often need to be updated and refined, which calls for incremental abilities. The proposed technique, embedded in a prototypical version of the document processing system DOMINUS, using the incremental first-order logic learner INTHELEX, revealed good performance in real-world experiments.

Stefano Ferilli · Teresa M.A. Basile · Nicola Di Mauro · Floriana Esposito
Dipartimento di Informatica – University of Bari (Italy)
e-mail: `{ferilli,basile,ndm,esposito}@di.uniba.it`

1 Introduction

The current spread of documents available in digital format raised the need of effective retrieval techniques based on their content. Since manual indexing is infeasible due to the amount of documents to be handled and to the subjectivity of experts' judgements, automatic techniques are the obvious solution. In particular, a key factor for such techniques to be successful is represented by the ability of properly identifying and understanding the structure of documents, in order to focus on the most significant components only. 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 the literature. However, they are mostly based on statistical and numerical approaches that may fail in classification and learning, being not able to deal with the lack of a strict layout regularity in the variety of documents available. On the other hand, on suitable descriptions of the document layout, Machine Learning techniques can be applied to automatically infer models of classes of documents and of their components. Indeed, organizing the documents on the grounds of the knowledge they contain is fundamental for being able to correctly access them according to the user's particular needs. For instance, in the scientific papers domain, in order to identify the subject of a paper and its scientific context, an important role is played by the information available in components such as Title, Authors, Abstract and Bibliographic references.

Thus, 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. To this aim, in this work we propose the application of Machine Learning techniques to infer models of correction rules for wrong document layouts from sample corrections performed by expert users, in order to automatically apply them to future incoming documents. This task requires a first-order logic representation, because the structure of a document is not fixed, and the corrections typically depend on the relationships of the wrong components with the surrounding ones. Moreover, as a consequence of the continuous flow of new and different documents, the learned models often need to be updated and refined, which calls for incremental abilities of the system. Indeed, the layout correction setting prevents the possibility of defining and fixing the number of correction typologies since the beginning of layout analysis step, as it is not possible to foresee how many and what kinds of corrections have to be performed in order to obtain a satisfactory document layout. Hence the need of incremental abilities of the approach in order to be able to deal with totally new layout correction instances.

This chapter proposes a technique that was actually embedded in a prototypical version of the document processing system DOMINUS, using the incremental first-order logic learner INTHELEX. Experiments in a real-world task confirmed the good performance of the proposed solution. The chapter is organized as follows: Sections 2 and 3 introduce the background in which the proposed solution is intended to work; Sections 4 and 5 present the details of the proposal; Section 6 discusses the experimental evaluation of the technique; lastly, Section 7 concludes the work.

2 Related Work

Document Image Understanding (*DIU*) is the process aimed at transforming the informative content of a (paper or digital) document into an electronic format outlining its logical content. DIU strongly relies on a preliminary Document Image Analysis process, whose goal is to discover the geometrical and logical layout structure of the document. Specifically, the geometric layout analysis aims at producing a description of the geometric structure of the document, while the logical layout analysis aims at identifying the different logical roles of the detected regions (titles, paragraphs, captions, headings) and the relationships among them. In this work we focus our attention on the geometric layout analysis step, with a particular emphasis on the ability for a document analysis system to automatically fix the wrong behaviour of this phase.

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. Specifically, it is possible to distinguish text segmentation approaches, which analyse the document in order to extract and segment text. The textual part is divided into columns, paragraphs, lines, and words in order to reveal the hierarchical structure of the document. Page segmentation approaches aim at partitioning the document into homogeneous regions. They can be grouped into different typologies according to the technique they adopt: smoothing/smearing [1], projection profile analysis [2, 3, 4, 5, 6, 7], texture-based or local analysis [8, 9], and analysis of the background structure [10, 11]. On the other hand, there exist the Segmentation/Classification mixed approaches. These hybrid approaches do not clearly separate the segmentation step from the classification one. The techniques they exploit are based on connected component analysis [12, 13] and texture or local analysis [14, 15]. Finally, the block classification approaches aim at labelling regions previously extracted in a block segmentation phase. Most of them are based on feature extraction and linear discriminant classifiers [1, 3, 7, 16].

All these approaches can be viewed as relying on two basic operations: split and merge. Indeed, they all 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, bottom-up or hybrid approach to the page decomposition step.

One of the most representative top-down approaches is the recursive X–Y cut method [5] that relies on projection profiles to cut a textual region into several subregions. However, this approach has some difficulty in dealing with text regions that lack a fully extended horizontal or vertical cut. In the same way, approaches that exploit maximal white rectangles [17] or white streams [7] may also fail to find white margins that are large enough. For this reason, a multi-scale analysis method that examines a document at various scales was proposed [18].

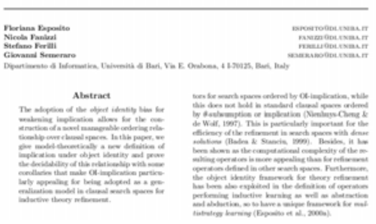
Significant examples of the bottom-up behavior are the document spectrum method [19], the minimal-cost spanning tree method [13], and the component-based algorithm [20]. The underlying idea of these methods is to build basic blocks based on the distances between connected components. In particular, [21] used the spanning tree as a pre-classifier to gather connected components into sub-graphs. In this approach, one crucial step involves cutting away a vertical sub-graph that has been wrongly merged into a horizontal text line, or vice versa. Some empirically estimated heuristics are exploited to solve this problem. Other authors developed a rule-based bottom-up method [22] in which a set of rules is encoded, that allows to merge connected components in text lines. Specifically, the rules are based on the concept of nearest-neighbour connect-strength, which varies according to the size similarity, distance, and offset of the components. In [23] a hybrid segmentation method is proposed, that partitions a document into blocks based on field separators and white streams, and then merges the components of each block into text lines.

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, where split or merge operations are performed using estimated parameter values, are present in the literature [24, 25]. 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 [26] and to classify text regions as headline, main text, etc. [27, 28] or even to learn split/merge rules in order to carry out the corresponding operations and/or correction [29, 30].

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.

3 Preliminaries

Based on the ODA/ODIF standard, any document can be progressively partitioned into a hierarchy of abstract representations, called its *layout structure*. Here we describe an approach, named DOC (Document Organization Composer) [31], for discovering a full layout hierarchy in digital documents based primarily on layout



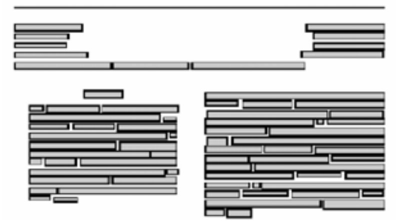
Floriana Esposito ESPOSITO@DI.UNIBA.IT
Nicola Fanizzi FANIZZI@DI.UNIBA.IT
Stefano Ferilli FERILLI@DI.UNIBA.IT
Giovanni Semeraro SEMERARO@DI.UNIBA.IT
Dipartimento di Informatica, Università di Bari, Via E. Orabona, 4 I-70125, Bari, Italy

Abstract

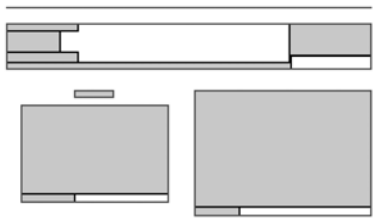
The adoption of the *object identity* bias for weakening implication allows for the construction of a novel manageable ordering relationship over clausal spaces. In this paper, we give model-theoretically a new definition of implication under object identity and prove the decidability of this relationship with some corollaries that make OI-implication particularly appealing for being adopted as a generalization model in clausal search spaces for inductive theory refinement.

tors for search spaces ordered by OI-implication, while this does not hold in standard clausal spaces ordered by θ-subsumption or implication (Nienhuys-Cheng & de Wolf, 1997). This is particularly important for the efficiency of the refinement in search spaces with *dense solutions* (Badea & Stanciu, 1999). Besides, it has been shown as the computational complexity of the resulting operators is more appealing than for refinement operators defined in other search spaces. Furthermore, the object identity framework for theory refinement has been also exploited in the definition of operators performing inductive learning as well as abstraction and abduction, so to have a unique framework for *multistrategy learning* (Esposito et al., 2000a).

a



b



c

Fig. 1 Layout analysis steps on the original document (a): preprocessing for world/line blocks aggregation (b) and final high-level layout structure (c).

information. The layout analysis process, whose main steps are shown in Figure 1, 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. Specifically, such a learning task was cast to a Multiple Instance Problem and solved by exploiting the kernel-based algorithm proposed in [32].

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 [33]. 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*') $C = \{r_0, \dots, r_n\}$, 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 representation based on the following data structures:

Rectangle represented by the coordinates of its *L*eft-*T*op and *R*ight-*B*ottom corners in the plane, i.e. (x_L, y_T, x_R, y_B) respectively;

Bound a pair (r, O) made up of a Rectangle r and a Set of obstacles O, each of which is in turn a Rectangle overlapping r;

Priority Queue of Bounds Q, whose elements are organized according to the area of r and providing direct access to the element having the maximum value for the area of r;

Set of Rectangles B, that collects the pieces of background as long as they are identified.

Elements are iteratively extracted from the queue: if the set of obstacles corresponding to the extracted element is empty, then it represents the maximum white rectangle still to be discovered, so it is saved into B and added as an additional obstacle to all bounds in the current queue to which it overlaps; otherwise one of its obstacles is chosen as a *pivot* and the contour is consequently split into four regions (those resting above, below, to the right and to the left of the pivot), some of which partially overlap. Each such region, along with the obstacles that fall in it, represents a new structure to be inserted in the priority queue. In the end, computing the complement of the areas in B would yield the document content blocks. The described procedure is formally specified in Algorithm 1.

Algorithm 1. Maximal White Rectangle

```
 1: $B \leftarrow \emptyset$
 2: $Q \leftarrow \{(r_b, C)\}$
 3: while $Q \neq \emptyset$ do
 4:     $(r, O) \leftarrow$ dequeue(Q)
 5:     if $O = \emptyset$ then
 6:         $B \leftarrow B \cup \{r\}$
 7:         Add $r$ as a new obstacle to all elements in $Q$ overlapping it
 8:     else
 9:         $p \leftarrow$ pivot($O$)
10:         $Q \leftarrow Q \cup \{(\ (r.x_L, r.y_T, r.x_R, p.y_T)\ ,\ \{b \in O | b \cap r_A \neq \emptyset\}\ )\}$
11:         $Q \leftarrow Q \cup \{(\ (r.x_L, p.y_B, r.x_R, r.y_B)\ ,\ \{b \in O | b \cap r_B \neq \emptyset\}\ )\}$
12:         $Q \leftarrow Q \cup \{(\ (r.x_L, r.y_T, p.x_L, r.y_B)\ ,\ \{b \in O | b \cap r_L \neq \emptyset\}\ )\}$
13:         $Q \leftarrow Q \cup \{(\ (p.x_R, r.y_T, r.x_R, r.y_B)\ ,\ \{b \in O | b \cap r_R \neq \emptyset\}\ )\}$
14: return $B$
```

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, such as single blocks filled with the same kind of content and rectangular *frames* (meaningful collections of objects completely surrounded by white space that may be made up of many blocks of the former type). 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. Indeed, this is the point in which all the useful white spaces in the document, e.g. those between columns and sections, have been identified. Such a consideration is generally valid for all the documents except for those having a scattered appearance. It is worth noting that this value is reached very early (around 1/4 of the steps needed by the algorithm to reach its natural end), and before the size of the structure containing the blocks waiting to be processed starts growing dramatically, thus saving lots of time and space resources.

Additional improvements are:

- choosing as a pivot a 'side' block (i.e., the top or bottom or leftmost or rightmost block), or even better a 'corner' one (one which is at the same time both top or bottom and leftmost or rightmost), which results in a quicker retrieval of the background areas with respect to choosing it at random or in the middle of the bound;
- considering horizontal/vertical lines in the layout as a natural separators, and hence adding their surrounding white space to the background before the algorithm starts;
- discarding any white block whose height or width is below a given threshold as insignificant (this should avoid returning inter-word or inter-line spaces).

In the end, each black block should correspond to a section or column in the page, depending on the layout detail level. The lower the threshold, the more white spaces should be identified. Starting from these blocks, the frames can be found.

4 Learning Layout Correction Theories

The strategy for automatically assessing a threshold to stop the background retrieval loop allows to immediately reach a layout. Often the identified layout is already good for many further tasks of document image understanding. Nevertheless, such a threshold is clearly a tradeoff 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. A first, straightforward way to perform this adjustment is allowing the user to take the loop some additional steps forward with respect to the stop condition, adding more background, or some steps backward, removing the most recently found background. However, there are cases in which the greedy technique it implements does not guarantee an optimal outcome. For instance, retrieving useful (e.g., those that concur to separate different target frames in the document layout) but very small pieces of background (possibly due to excessive fragmentation operated by the algorithm) might require, using the normal priority-based behavior, to preliminary retrieve several meaningless ones as a side-effect. Even worse, some pieces of background might not be retrieved at all, being smaller than the minimum size threshold. To handle these cases, a tool was provided that allows the user to directly point out useful background fragments that were not yet retrieved from the queue and add them explicitly, or, conversely, to select useless ones that were erroneously retrieved and remove them from the final layout. Let us call these types of manual intervention '*white forcing*' and '*black forcing*', respectively. In the rest of the chapter, the following groups of terms will be considered as synonyms, referred to the document layout:

- white, background;
- black, foreground, content;
- contour, area, rectangle.

4.1 From Manual to Automatic Improvement of the Layout Correction

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:

1. if the contour is not empty, it is split and the resulting fragments are enqueued;
2. if the contour is empty and fulfils the constraints, it is added to the list of white areas;
3. 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 behavior as a consequence of his choices. First of all, the white areas that were discarded because too small must be stored in a list. This would allow the user to retrieve any white space in the document page. Thus, since in general, due to the stop criterion, the algorithm terminates much before the queue is completely emptied, the following structures will be available as its outcome:

- a queue Q of the contours yet to be examined, some of which actually white, others to be further split;
- a list B of the white areas (background blocks) retrieved thus far;
- a list D of the discarded white areas (background blocks) deemed as insignificant because not satisfying the constraints.

Now, the interactive extension allows the user to make the algorithm perform some more steps forward or backward, causing a corresponding movement of blocks among these three structures. For instance, when a step forward is requested, Q

is processed until the first white contour that satisfies the constraints is found and put in B. As already noticed, this functionality is still not sufficient, because significant but very small white rectangles might require several steps forward to be retrieved, causing the previous retrieval of several insignificant ones in the meantime. In the worst case, such significant rectangles might not be retrieved at all (they would be discarded if not satisfying the constraints), preventing a correct layout to be returned. In these cases, the user would directly 'force' the retrieval or removal of specific blocks from the document layout, by pointing with the mouse the corresponding area. If he wants to force the retrieval of a background piece (white forcing), all the white blocks in Q and D are scanned to find one underlying the pointed spot, and in case it is found it is added to the background B (even if it does not satisfy the requirements). A technical trick is needed to properly manage these blocks in case of backward/forward steps by the user. Indeed, if put back into the queue by taking a step backward, the blocks would be placed at the bottom, and hence a subsequent step forward would not retrieve them immediately, as expected (being considered as the inverse of the backward step). To avoid this, they are assigned as priority the area of the rectangle placed in that moment at the top of the queue, instead of their actual area. This also solves the problem of losing meaningful whites. Conversely, the 'black forcing' functionality allows the user to retrieve and remove a white block from B in order to include it in a frame, placing it in D. Both these operations can be undone, this way restoring the original algorithm behavior.

From the above discussion, 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 logfile 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. For instance, non-forced white blocks can be considered as negative examples for '*black forcing*', while the discarded blocks and the non-forced blocks in the queue can be assumed to be negative examples for '*white forcing*'. 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.

4.2 Tool Architecture

The whole procedure of preliminary layout analysis and automatic layout correction, endowed with learning functionality to improve future performance, was implemented in the Java language. A Document is considered as an aggregate of Pages, each of which specifies six lists of components, one for each of the following kinds:

Box	a fragment of text as extracted from the source file (possibly a single letter or just a fragment of word);
Word	a set of overlapping or adjacent, and co-linear, Boxes;
Line	a set of co-linear Words belonging to the same text column;
Stroke	a graphical line in the document;
Fill	a solid rectangle in the document;
Image	a raster image extracted from the source document, of an aggregation of overlapping graphical items (such as Fills and Strokes);

and separately undergoes the layout analysis process as described before. This process results in two kinds of aggregate structures, namely blocks and frames, represented as their minimum bounding box. Boxes, words, lines, images, strokes, fills, blocks, frames and pages are described as generic rectangles, according to the coordinates of their top-left and bottom-right corners, plus a unique identifier. This way, the computation of areas, distances, overlappings etc. can be easily carried out as geometrical operations. More specifically, the following main functionality is provided for:

stepForward	applies a single step of the layout analysis algorithm, by processing the queued boundaries until the first white is found;
findLayout	repeatedly applies 'stepForward's until the pre-specified threshold is reached, and returns a contour representing the page, containing a set of background blocks ('whites') retrieved;
findInverse	returns a contour representing the page, containing a set of content blocks ('blacks') obtained as the complement of the background;
stepBackward	removes the last white found from the current layout reconstruction;
forceWhite	forces the retrieval of a white block, overriding the normal behavior of the findLayout algorithm;
forceBlack	forces the removal of a specific white block previously found, in order to consider it as content ('black');
undoForceWhite	withdraws the forcing of a white block;
undoForceBlack	withdraws the forcing of a black block.

Additional service functionality allows, given a coordinate of the page, to scan the queue in order to retrieve an underlying queued boundary, or to search for an underlying white block from the current background.

DOC also maintains the log of manual corrections performed by the expert user. After he acknowledges the final layout, a translation functionality is activated to transform all manual applied corrections for black and white blocks into positive examples, and to produce a corresponding set of negative examples, for the two classes 'force_white' and 'force_black'. Hence, on such a log file of manual corrections, a typical problem of supervised inductive learning is set up and, since the relationships between different objects are significant, a first-order logic description language is used to represent these logs. At this stage, a first-order logic learner is exploited. Specifically, here we adopt INTHELEX that was already successfully exploited for document image understanding tasks [34] and hence turns out to be a natural candidate for application. Indeed, the incremental characteristic of the system allows to exploit each manual correction to refine the learned theory, and hence contributes to progressively optimize the system behavior and avoid further user intervention.

4.3 The Learning System

INTHELEX is an Inductive Logic Programming [35] system that learns hierarchical first-order logic theories from positive and negative examples. It is fully incremental (in addition to the possibility of refining previously generated hypotheses/definitions, learning can also start from an empty theory), and adopts a representation language that ensures effectiveness of the descriptions and efficiency of their handling, while preserving the expressive power of the unrestricted case [36]. It can learn simultaneously multiple concepts/classes, possibly related to each other, and it guarantees validity of the learned theories on all the processed examples.

The system is able to exploit feedback on performance to activate the theory revision phase, as described in the following. An initial theory is set up for the target concepts, learned by the system from a previous set of examples selected from the environment and classified by an expert, or provided directly by the expert, or even empty. Subsequently, such a theory can be applied to new available observations, producing a decision. If an oracle is available, that compares such a decision to the correct one, whenever the prediction is incorrect, the cause of such a wrong decision can be automatically pointed out by the system and the proper kind of correction applied by a theory revision process. In this way, it is able to incrementally modify incorrect theories according to a data-driven strategy.

Specifically, when a positive observation is not covered, a revision of the theory to restore its completeness is performed as follows: replacing a rule in the theory with one of its least general generalizations against the problematic observation; adding a new rule to the theory, obtained by properly turning constants into variables in the problematic example; adding the problematic observation as a positive exception. On the other hand, when a negative observation is covered, the system revises the theory to restore consistency by performing one of the following actions: adding *positive* information able to characterize all the past positive observations (and exclude the problematic one) to the rule that covers the example; adding *negative* information to discriminate the problematic observation from all the past positive ones to the rule that covers the problematic observation; adding the problematic observation as a negative exception.

Another peculiarity in the learning system is the embedding of multistrategy operators that may help in solving the theory revision problem. Induction, Deduction, Abduction and Abstraction were integrated according to the Inferential Theory of Learning theoretical framework [37]. For the present study, abstraction, in particular, is a precious support to tackle the complexity of the domain and of the related descriptions. It is a pervasive activity in human perception and reasoning, and aims at removing superfluous details from the description of both the examples and the theory. Thus, the exploitation of abstraction results in the shift from the language in which the theory is described to a higher level one. According to the framework proposed in [38], in INTHELEX abstraction takes place by means of a set of operators that replace a number of components by a compound object, or decrease the granularity of a set of values, or ignore whole objects or just part of their features, or neglect the number of occurrences of some kind of object.

5 Description Language

Now let us turn to the way in which the manual corrections are to be described. The assumption is that the user changes the document layout when he considers that the proposed layout is wrong (by observing it *before* applying the correction), then he forces a specific block because he knows the resulting effect on the document (he foresees the situation *after* the correction) 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.

Each learning example is represented as a first-order logic clause $H\ :-\ B$, whose head reports whether the description in the body represents a positive or negative example for the forcing of a black or white block. Positive examples are denoted by the following predicates:

- `force_white(forced_block, document)`
- `force_black(forced_block, document)`

while negative examples are expressed as negations thereof. The clause body is built on the set of predicates reported in Table 1. It contains information about the page in which the correction took place, i.e. horizontal/vertical size and position in the overall document (whether it is at the beginning, in the middle or at the end of the document, and specifically whether it is the first or last one), furthermore it describes the forced block and the layout situation both before and after the correction. Specifically, for a given correction *id_correction*, to denote these two moments, corresponding identifiers, *id_correction*_before and *id_correction*_after, are generated and introduced by two specific literals, respectively:

- `before(`*`idCorrection`*`,`*`idCorrection`*`_before)`
- `after(`*`idCorrection`*`,`*`idCorrection`*`_after)`

Table 1 First-order logic descriptors for layout correction observations: Attributes (bold) and Relations (italic) to be applied to entities (Document, Page, Block, Frame)

Correction identification	
before(idCorrection,idCorrection_before)	identifies the layout description before correction
after(idCorrection,idCorrection_after)	identifies the layout description after correction
Page General Information	
page(idDocument,idPage)	correction applied to page idPage in document idDocument
page_height(idPage,h)	h is the height of page idPage
page_width(idPage,w)	w is the width of page idPage
first_page(idPage)	idPage is the first page of the document
last_page(idPage)	idPage is the last page of the document
first_pages(idPage)	idPage belongs to the first 1/3 pages of the doc
middle_pages(idPage)	idPage belongs to the middle 1/3 pages of the doc
last_pages(idPage)	idPage belongs to the last 1/3 pages of the doc
Rectangle (Block or Frame) General Information (% wrt the page dimensions)	
pos_x(idRectangle,x)	horizontal position x of the centroid of rectangle
pos_y(idRectangle,y)	vertical position y of the centroid of rectangle
width(idRectangle,w)	width w of rectangle idRectangle
height(idRectangle,h)	height h of rectangle idRectangle
Rectangle Typology	
frame(idPage,idFrame)	for each frame idFrame that touches or overlaps the forced block
block(idPage,idBlock)	for each block idBlock that touches or overlaps the forced block
type(idRectangle,Type)	Type $\in$ {text, line, image, mixed}
Relationships between Rectangles (Blocks or Frames)	
belongs(idBlock,idFrame)	block idBlock belongs to frame idFrame
overlaps(idForcedBlock,idBlock, p)	p is the percentage of overlapping between the two blocks idForcedBlock and idBlock
touches(idForcedBlock, idRectangle)	rectangle idRectangle touches (but does not overlap) the forced block idForcedBlock
overlap_part_N(idRectangle1, idRectangle2)	spatial relation ($N \in \{1,\ldots,25\}$) between idRectangle1 and idRectangle2 according to the 25-plane model; specifically, between a block or frame and the forced block, or between two frames involved in the correction being described (each of which touches or overlaps the forced block), or between two blocks in the same frame (each of which touches the forced block)

The description of each of the two situations (before and after the correction) is based on literals expressing the page layout. While the predicates, on which such literals are built, are mostly the same that can be used for document image understanding purposes, for the specific task aimed at learning correction rules it is

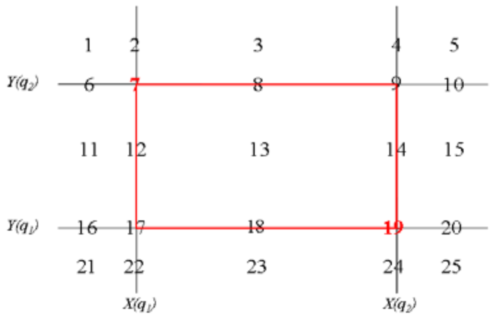


Fig. 2 Partition of the plane with respect to a rectangle according to [39]

sufficient to express only a subset of the whole layout, and specifically the neighborhood of the forced block that is considered as the most significant and useful to understand why that block is being forced. In our case, the focus was put on the blocks and frames surrounding the forced block, and, among them, only on those touching or overlapping the forced block. The relationships between these components 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, that touch or overlap the forced block. Furthermore, for each frame or block that touches the forced block, a literal specifying that they touch is introduced. Finally, for each block of a frame that overlaps the forced block, the percentage of overlapping is reported.

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 [39] for representing the spatial relationships among the blocks/frames was considered. Specifically, according to such a model, once a rectangle is fixed, its plane is partitioned in 25 parts (as shown in Figure 2) and its spatial relationships to any other rectangle in the plane can be specified by simply listing the parts with which the other rectangle overlaps. From this basic representation, higher-level topological relations [40, 39], such as closeness, intersection and overlapping between rectangles can be automatically derived using deductive capabilities.

For instance, the following fragment of background knowledge could be provided to the learning system:

$$\begin{array}{l} overlap\_part\_14(B1,B2) \land \neg overlap\_part\_13(B1,B2) \Rightarrow touch(B1,B2) \\ overlap\_part\_17(B1,B2) \land \neg overlap\_part\_13(B1,B2) \Rightarrow touch(B1,B2) \\ overlap\_part\_18(B1,B2) \land \neg overlap\_part\_13(B1,B2) \Rightarrow touch(B1,B2) \\ overlap\_part\_19(B1,B2) \land \neg overlap\_part\_13(B1,B2) \Rightarrow touch(B1,B2) \end{array}$$

and, given a description involving two blocks $b1$ and $b2$, and including a literal $overlap\_part\_14(b2,b1)$, but not a literal $overlap\_part\_13(b2,b1)$, it would be able to automatically recognize that $touch(b2,b1)$.

Finally, each involved frame or block is considered as a rectangular area of the page, and described according to the following parameters:

- horizontal and vertical position of the rectangle centroid with respect to the top-left corner of the page (coordinates expressed as percentages of the page dimensions),
- height and width of the rectangle (expressed as percentages of the page dimensions), and
- content type (text, graphic, line).

Note that, as regards the relationships between frames and blocks, since only those frames that touch or overlap the forced block are taken into account, if the forced block touches or overlaps just one frame no instance of such relationships will be included; the same holds for the blocks in a frame that touch or overlap the forced block. This often happens when the correction significantly changes the layout structure, modifying the number of blocks or frames. To clearly distinguish the various corrections, they are identified by the document identifier, followed by the page number and lastly by the progressive block number in that page.

The example depicted in Figure 3 shows a case in which the user must force a white block (*b*2 in figure) to become black because it is interpreted as a spacing. The reference frame (*f*1 in figure) contains two other blocks, one of which (*b*1) overlaps the rectangle to be forced (*b*2), while the other (*b*3) touches it. After the correction, the situation is as follows: there are two frames (*f*2 and *f*3) that touch the forced rectangle, both containing a block (*b*4 and *b*5 respectively) that touches it. The corresponding clause is:

```
[not(force_black(b2,d1),force_white(b2,d1)]:-
    ... general information of the block to force b2
    before(d1,d1_before), page_1(d1_before,d1_before_p1),
    ... general information of the page
    frame(d1_before_p1,f1), overlaps(b2,f1,100%),
    .... general information of the frame f1
    // spatial relationships between blocks f1 and b2
    overlap_part_2(f1,b2), overlap_part_3(f1,b2),
    overlap_part_4(f1,b2), overlap_part_5(f1,b2),
    overlap_part_7(f1,b2), overlap_part_8(f1,b2),
    overlap_part_9(f1,b2), overlap_part_10(f1,b2),
    overlap_part_12(f1,b2), overlap_part_13(f1,b2),
    overlap_part_14(f1,b2), overlap_part_15(f1,b2),
    overlap_part_17(f1,b2), overlap_part_18(f1,b2),
    overlap_part_19(f1,b2), overlap_part_20(f1,b2),
    overlap_part_22(f1,b2), overlap_part_23(f1,b2),
    overlap_part_24(f1,b2), overlap_part_25(f1,b2),
    block(d1_before_p1,b3), belongs(b3,f1), touches(b2,b3),
    .... general information of the block b3
    block(d1_before_p1,b1), belongs(b1,f1),overlaps(b2,b1,10%),
    .... general infomation on the block b1
    ... spatial relationships between blocks b3 and b2
    ... spatial relationships between blocks b1 and b2
```

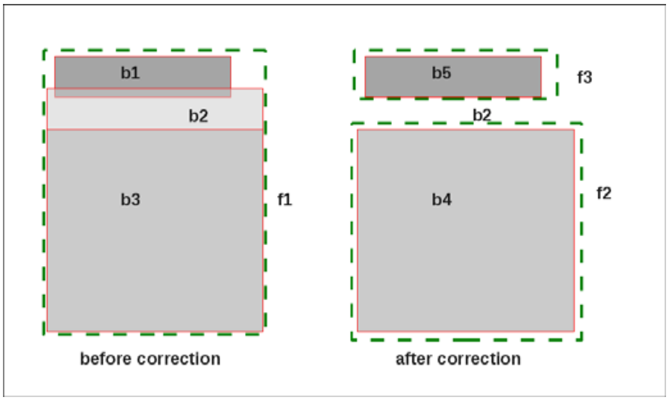


Fig. 3 Example of a situation before and after the correction

```
... spatial relationships between blocks b3 and b1
after(d1,d1_after), page_1(d1_after,d1_after_p1),
frame(d1_after_p1,f2),  touches(b2,f2),
.... general infomation on the frame f2
block(d1_after_p1,b4), belongs(b4,f2),  touches(b2,b4) ,
.... general infomation on the block b4
frame(d1_after_p1,f3),  touches(b2,f3),
.... general infomation on the frame f3
block(d1_after_p1,b5), belongs(b5,f3),   touches(b2,b5) ,
 .... general information on the block b5
... spatial relationships between blocks f2 and b2
... spatial relationships between blocks b4 and b2
... spatial relationships between blocks f3 and b2
... spatial relationships between blocks b5 and b2
... spatial relationships between blocks f2 and f3
```

6 Experiments

The proposed description language was used to run two experiments 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: 'force white' (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 'force black' (corresponding to the fact that a white rectangle found by the lauyout analysis algorithm must, instead, be discarded). In both cases, a 10-fold cross-validation technique was exploited to obtain the training and test sets. The experiments were run on a PC endowed with WindowsXP running on an Intel 2.4 GHz Core 2 Duo Processor and 2 GB RAM. INTHELEX was set so as to force each generalization to preserve not less than 40% of the original clause description and no more that 70%

thereof (computed as the number of literals in the body), in order to limit the search space and, as a consequence, the computational requirements.

The experimental dataset concerned the corrections applied to obtain the correct layout on about one hundred documents (specifically, papers published in scientific journals and conference proceedings), evenly distributed in four categories (ECAI, Elsevier, ICML, Springer-Verlag Lecture Notes). According to the strategy described above, the examples concern significant background blocks that were not retrieved ('white forcing') or useless white blocks erroneously considered as background ('black forcing') by the basic layout analysis algorithm. Since the layout analysis algorithm already treats isolated graphic lines as separators, it was necessary to force white blocks around such lines. The experimental evaluation was summarized in tables organized as follows. For each fold (specified in column *Fold*), the following figures are reported: the number of corresponding positive (*PosEx*) and negative (*NegEx*) training examples, the number of clauses in the learned theory (*NewCl*), the number of generalizations performed (*Lgg*), the number of positive exceptions introduced in the theory (*Pexc*), the number of positive (*Plit*) and negative (*Nlit*) literals added to specialize the theory, the number of negative exceptions introduced (*Nexc*), runtime in seconds (*Time*), predictive accuracy of the learned theories on the corresponding test sets (% in percentage), the ratio between the positive examples covered by the theory and the total number of positive examples ($TP = Pcov/TotExPos$) and, lastly, the ratio of rejected negative examples over the total number of negative examples ($TN = NNCov/TotExNeg$).

For the first dataset the layout correction activity resulted in a set of 786 examples of block correction, of which 263 for 'force white' and 523 for 'force black'. Positive examples for 'force white' were considered as negative for 'force black' and *vice versa*, this way exploiting the whole dataset. Thus, each single correction was interpreted from two perspectives: obviously 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 head for a block forced_block forced white is as follows:

```
[
not(force_black(forced_block, document)),
force_white(forced_block, document)
]
```

while for a forced black is:

```
[
not(force_white(forced_block, document)),
force_black(forced_block, document)
]
```

Table 2 reports the results for concept 'force white'. Given the amount of available examples in the dataset, the experimental outcome reveals good predictive accuracy of the learned theories, that is 98,47% on average, never falling below 94.87%, and reaching 100% in 4 cases out of 10. Also runtimes are satisfactory: an average of 175.13 seconds (i.e., about 3 minutes), mostly due to a single problematic case that required 790.17 seconds (about 13 minutes) for accomplishment. INTHELEX added 1.4 positive literals and 1.5 negative ones on average during the specialization steps. The number of generalizations performed (16 on average) and the small number of clauses for each theory (5.4 on average) are good, because they indicate that the positive examples shared common features. A possible weakness is the number of negative exceptions (5 in the case of fold fb04), but a possible explanation is that the parameter that requires each generalization to preserve at most 70% of the description length tends to yield too general theories. Lastly, the ratios of correctly classified test examples (True Positives, TP, and True Negatives, TN, respectively) reveals that the various theories never failed on more than two negative examples, indicating a cautious behavior, as desired. In general, it might happen that too cautious theories cover too few positive examples, and hence might be useless, but the number of covered test examples (Pcov) shows this is not the case (the theories miss at most 4 thereof). In particular, the theory learned in fold fb05 might be selected as the best theory to be exploited, having predictive accuracy of 100%, just 4 clauses and no exception.

Table 2 Experimental outcomes for concept 'force white'

Fold	TP	TN	PosEx	NegEx	NewCl	Lgg	Pexc	Plit	Nlit	Nexc	Time	%
fb01	27/27	52/53	236	470	6	17	0	2	0	0	120.54	98.75
fb02	26/27	53/53	236	470	7	16	0	2	4	0	271.89	98.75
fb03	27/27	53/53	236	470	5	17	0	1	1	0	83.71	100
fb04	26/26	50/52	237	471	6	21	0	2	6	5	790.17	97.44
fb05	26/26	52/52	237	471	4	10	0	2	1	0	36.87	100
fb06	22/26	52/52	237	471	6	19	0	0	0	0	171.74	94.87
fb07	24/26	51/52	237	471	6	19	0	2	0	0	72.84	96.15
fb08	25/26	52/52	237	471	4	11	0	1	1	0	46.55	98.72
fb09	26/26	52/52	237	471	5	17	0	2	1	0	90.56	100
fb10	26/26	52/52	237	471	5	13	0	0	1	0	68.12	100
Average					5.4	16	0	1.4	1.5	0.5	175.13	98.47
Mean Dev.					0.97	3.59	0	0.84	1.96	1.58	226.98	1.79
Max					7	21	0	2	6	5	790.17	100
Min					4	10	0	0	0	0	36.87	94.87

Table 3 reports the results for concept 'force black'. In this case, there were more positive examples than negative ones. The figures are again satisfactory, although slightly worse than those obtained for the previous concept. Indeed, the predictive accuracy of the various theories amounts to 97.82% on average, never falls below 94.87% and only in two cases reaches 100% (fn02 and fn03). Runtimes are worse than those of the previous experiment (394.41 seconds, i.e. about 6.5 minutes, on average). The number of generalizations carried out (17.6 on average) is quite similar for the previous concept. The number of negative exceptions is high (up to 13 in fold fn09), which means that there are negative examples (i.e., positive examples for 'force white') very similar to the positive ones, so that INTHELEX is not able to distinguish between them. However, it is encouraging that the best theories for predictive accuracy (fn02 and fn03) has very few (2 and 1 respectively). Lastly, by observing the ratio between covered examples and total examples during the test phase (Pcov and NNCov), one can note that the number of uncovered examples is quite stable. The best theory, among those produced by the ten folds, to be chosen for use seems to be that of fold fn03, that has 100% accuracy, just 2 clauses and one negative exception.

Table 3 Experimental outcomes for concept 'force black'

Fold	TP	TN	PosEx	NegEx	NewCl	Lgg	Pexc	Plit	Nlit	Nexc	Time	%
fn01	52/53	27/27	470	236	4	12	0	2	0	0	85.62	98.75
fn02	53/53	27/27	470	236	3	13	0	6	2	2	235.04	100
fn03	53/53	27/27	470	236	2	8	0	2	2	1	105.34	100
fn04	51/52	25/26	471	237	5	20	0	4	6	7	618.49	97.44
fn05	51/52	25/26	471	237	8	30	0	7	1	0	309.46	97.44
fn06	49/52	26/26	471	237	6	15	0	5	7	4	638.58	96.15
fn07	50/52	24/26	471	237	7	26	0	4	0	1	257.05	94.87
fn08	51/52	26/26	471	237	5	21	0	8	8	2	820.33	98.72
fn09	51/52	25/26	471	237	2	13	0	3	4	13	462.44	97.44
fn10	51/52	25/26	471	237	4	18	0	1	0	0	411.78	97.44
Average					4.6	17.6	0	4.2	3	3	394.41	97.82
Mean Dev.					2.01	6.79	0	2.3	3.06	4.14	241.88	1.61
Max					8	30	0	8	8	13	820.33	100
Min					2	8	0	1	0	0	85.62	94.87

The outcomes of the first experiment suggest that the description language proposed and the way in which the forcings are described are effective to let the system learn rules 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. Recall that, 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 white blocks that can be forced to black, while the other two contain rectangles that might be forced to 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 anymore to consider each white block forcing as a counterexample for black forcing and *vice versa*, but to ensure that all learned rules are 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 'force black' (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 'force white' (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 categories (ICML, ECAI, Elsevier, Springer-Verlag Lecture Notes). Specifically, the new dataset included 113 positive and 840 negative examples for 'force black', and resulted in the performance reported in Table 4. The predictive accuracy was improved with respect to the previous experiment, reaching 99.16% on average, with several folds obtaining 100%, which is a very good result. The presence of very few negative exceptions (especially considering the number of examples), the presence of no more than 5 clauses in each theory along with the number of generalizations performed, suggest that the concept was properly learned and that the application of any of such theories to real cases may provide satisfactory results.

Table 4 Experimental outcomes for concept 'force black' (no neg ex)

Fold	TP	TN	PosEx	NegEx	NewCl	Lgg	Pexc	Plit	Nlit	Nexc	Time	%
fn01	12/12	84/84	101	756	4	12	0	0	1	0	77.81	100
fn02	12/12	83/84	101	756	3	8	0	1	1	0	62.25	98.96
fn03	12/12	84/84	101	756	4	11	0	1	0	0	57.73	100
fn04	9/11	84/84	102	756	3	11	0	0	2	0	241.60	97.89
fn05	11/11	84/84	102	756	3	9	0	0	0	1	88.52	100
fn06	9/11	83/84	102	756	3	11	0	1	0	2	153.15	96.84
fn07	11/11	84/84	102	756	4	12	0	0	1	0	94.16	100
fn08	10/11	84/84	102	756	2	7	0	0	0	4	69.38	98.95
fn09	11/11	84/84	102	756	5	14	0	4	1	0	264.99	100
fn10	11/11	83/84	102	756	3	8	0	1	0	1	164.81	98.95
Average					3.4	10.3	0	0.8	0.6	0.8	127.44	99.16
Mean Dev.					0.84	2.21	0	1.23	0.7	1.32	75.70	1.09
Max					5	14	0	4	2	4	264.99	100
Min					2	7	0	0	0	0	57.73	96.84

As to the concept 'force white', 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 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 'force black' concept. The experiment run on such a subset provided the results shown in Table 5. The number of clauses in each theory is significantly larger than that resulting from the 'force black' experiment suggesting that the white forcing examples have less common features. This is most likely due to the intrinsic complexity of the current concept, that is more complex than the previous one because many different situations may lead to forcing a white, differently from blacks that are typically forced to remove indentations. Another explanation might be that the lower bound of 40% set for generalizations during the learning phase causes the production of theories that are not very general. The predictive accuracy of the various theories is very encouraging (98.10% on average, with peaks of 98.95%). This is due to the reduction of the set of negative examples, indeed the various theories fail more on positive examples than on negative ones.

Table 5 Experimental outcomes for concept 'force white' (no neg ex)

Fold	TP	TN	PosEx	NegEx	NewCl	Lgg	Pexc	Plit	Nlit	Nexc	Time	%
fb01	10/11	84/84	90	759	6	13	0	0	4	3	166.63	98.95
fb02	9/10	85/85	91	758	9	20	0	2	2	1	249.31	98.95
fb03	7/10	84/85	91	758	9	22	0	0	5	0	343.20	95.79
fb04	8/10	84/85	91	758	7	14	0	1	6	2	375.85	96.84
fb05	10/10	82/84	91	759	7	10	0	4	8	3	284.60	97.87
fb06	9/10	84/84	91	759	10	19	0	1	3	1	335.25	98.94
fb07	9/10	84/84	91	759	13	17	0	4	2	1	519.40	98.94
fb08	10/10	83/84	91	759	8	14	0	0	3	2	163.16	98.94
fb09	8/10	83/84	91	759	7	18	0	2	5	1	213.40	96.81
fb10	9/10	84/84	91	759	9	13	0	2	2	5	525.50	98.94
Average					8.5	16	0	1.6	4	1.9	317.63	98.10
Mean Dev.					2.01	3.77	0	1.51	2	1.45	129.77	1.20
Max					13	22	0	4	8	5	525.50	98.95
Min					6	10	0	0	2	0	163.16	95.79

To confirm or reject this hypothesis, they were applied to the remaining negative examples previously discarded (9203), and the results (reported in Table 6) provide an ultimate confirmation of this: a predictive accuracy of 98.91% that, being obtained on so large a test set, is likely to correctly approximate the true behavior on real-world cases.

Table 6 Experimental outcomes for the test of concept 'force white' on the discarded 9203 negative examples

Fold	fb01	fb02	fb03	fb04	fb05	fb06	fb07	fb08	fb09	fb10	Av.	Max	Min
TN	9089	9122	9153	9124	9077	9111	9059	9135	9084	9069			
%	98.76	99.12	99.46	99.14	98.63	99	98.44	99.26	98.71	98.54	98.91	99.46	98.44

7 Conclusions

The huge amount of documents available in digital form and the flourishing of digital repositories raise the need of effective retrieval techniques based on their contents. Automatic techniques able to properly identify and understand the structure of documents in order to focus on the most significant components only seem to be the most suitable solution as manual indexing is infeasible due to the amount of documents to be handled. Hence, the quality of the layout analysis outcome is crucial because it determines and biases the quality of the next understanding steps.

However, due to the variety of document styles and formats to be processed, the layout analysis task is a non-trivial one and often the automatically found structure needs to be manually fixed by domain experts. In this work we proposed a tool, embedded in a prototypical version of the document processing system DOMINUS, able to use the steps carried out by the domain expert with the aim of correcting the outcome of the layout analysis phase. Specifically, the tool is able to infer rules for the layout correction to be applied to future incoming documents. It makes use of a first-order logic representation of the document structure because corrections often depend on the relationships of the wrong components with the surrounding ones. Moreover, the tool exploits the incremental abilities of the system INTHELEX as the continuous flow of new and different documents requires the learned models to be updated and refined. 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. Wong, K.Y., Casey, R.G., Wahl, F.M.: Document analysis system. IBM Journal of Reserch and Development 26, 647–656 (1982)
2. Nagy, G., Seth, S.: Hierarchical representation of optically scanned documents. In: Proceedings of the 7th International Conference on Pattern Recognition, pp. 347–349. IEEE Computer Society Press, Los Alamitos (1984)
3. Wang, D., Srihari, S.N.: Classification of newspaper image blocks using texture analysis. Computer Vision, Graphics, and Image Processing 47, 327–352 (1989)
4. Nagy, G., Seth, S., Viswanathan, M.: A prototype document image analysis system for technical journals. Computer 25, 10–22 (1992)
5. 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, 737–747 (1993)
6. Sylwester, D., Seth, S.: A trainable, single-pass algorithm for column segmentation. In: Procedings of International Conference on Document Analysis and Recognition, vol. 2, pp. 615–618. IEEE Computer Society Press, Los Alamitos (1995)
7. Pavlidis, T., Zhou, J.: Page segmentation and classification. CVGIP: Graphical Models Image Process. 54, 484–496 (1992)
8. Jain, A.K., Bhattacharjee, S.: Text segmentation using gabor filters for automatic document processing. Machine Vision and Applications 5, 169–184 (1992)

9. Tang, Y.Y., Ma, H., Mao, X., Liu, D., Suen, C.Y.: A new approach to document analysis based on modified fractal signature. In: Procedings of International Conference on Document Analysis and Recognition, vol. 2, pp. 567–570. IEEE Computer Society Press, Los Alamitos (1995)
10. Normand, N., Viard-Gaudin, C.: A background based adaptive page segmentation algorithm. In: ICDAR 1995: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 1, pp. 138–141. IEEE Computer Society Press, Los Alamitos (1995)
11. Kise, K., Yanagida, O., Takamatsu, S.: Page segmentation based on thinning of background. In: ICPR 1996: Proceedings of the International Conference on Pattern Recognition (ICPR 1996), vol. III, 7276, pp. 788–792. IEEE Computer Society Press, Los Alamitos (1996)
12. Wang, S.-Y., Yagasaki, T.: Block selection: a method for segmenting a page image of various editing styles. In: ICDAR 1995: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 1, pp. 128–133. IEEE Computer Society Press, Los Alamitos (1995)
13. Simon, A., Pret, J.-C., Johnson, A.P.: A fast algorithm for bottom-up document layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 273–277 (1997)
14. Sauvola, J., Pietikainen, M.: Page segmentation and classification using fast feature extraction and connectivity analysis. In: ICDAR 1995: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 2, pp. 1127–1131. IEEE Computer Society Press, Los Alamitos (1995)
15. Jain, A.K., Zhong, Y.: Page segmentation using texture analysis. Pattern Recognition 29, 743–770 (1996)
16. Shih, F.Y., Chen, S.S.: Adaptive document block segmentation and classification. IEEE Transactions on Systems, Man, and Cybernetics 26, 797–802 (1996)
17. Ittner, D., Baird, H.: Language-free layout analysis. In: ICDAR 1993: Proceedings of the Second International Conference on Document Analysis and Recognition, vol. 1, pp. 336–340. IEEE Computer Society Press, Los Alamitos (1993)
18. Lee, S.W., Ryu, D.S.: Parameter-free geometric document layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 23, 1240–1256 (2001)
19. O'Gorman, L.: The document spectrum for page layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 15, 1162–1173 (1993)
20. Liu, F.: A new component based algorithm for newspaper layout analysis. In: ICDAR 2001: Proceedings of the Sixth International Conference on Document Analysis and Recognition, pp. 1176–1180. IEEE Computer Society Press, Washington, DC, USA (2001)
21. Xi, J., Hu, J., Wu, L.: Page segmentation of chinese newspapers. Pattern Recognition 35, 2695–2704 (2002)
22. Chen, M., Ding, X., Liang, J.: Analysis, understanding and representation of chinese newspaper with complex layout. In: Proceedings of the 2000 International Conference on Image Processing (ICIP), pp. 90–93. IEEE Computer Society Press, Los Alamitos (2000)
23. 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 Press, Los Alamitos (1993)
24. 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, 1265–1278 (1997)

25. Chang, F., Chu, S.Y., Chen, C.Y.: Chinese document layout analysis using adaptive regrouping strategy. Pattern Recognition 38, 261–271 (2005)
26. 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, 92–96 (1997)
27. Dengel, A., Dubiel, F.: Computer understanding of document structure. International Journal of Imaging Systems and Technology 7, 271–278 (1996)
28. Laven, K., Leishman, S., Roweis, S.: A statistical learning approach to document image analysis. In: ICDAR 2005: Proceedings of the Eighth International Conference on Document Analysis and Recognition, pp. 357–361. IEEE Computer Society Press, Los Alamitos (2005)
29. Malerba, D., Esposito, F., Altamura, O., Ceci, M., Berardi, M.: Correcting the document layout: A machine learning approach. In: ICDAR 2003: Proceedings of the Seventh International Conference on Document Analysis and Recognition, pp. 97–103. IEEE Computer Society Press, Los Alamitos (2003)
30. Wu, C.C., Chou, C.H., Chang, F.: A machine-learning approach for analyzing document layout structures with two reading orders. Pattern Recogn. 41, 3200–3213 (2008)
31. 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. SCI, vol. 90, pp. 105–138. Springer, Heidelberg (2008)
32. Dietterich, T.G., Lathrop, R.H., Lozano-Perez, T.: Solving the Multiple Instance Problem with axis-parallel rectangles. Artificial Intelligence 89, 31–71 (1997)
33. Breuel, T.M.: Two geometric algorithms for layout analysis. In: Lopresti, D.P., Hu, J., Kashi, R.S. (eds.) DAS 2002. LNCS, vol. 2423, pp. 188–199. Springer, Heidelberg (2002)
34. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T.M.A., Di Mauro, N.: Incremental multistrategy learning for document processing. Applied Artificial Intelligence Journal 17, 859–883 (2003)
35. Muggleton, S., Raedt, L.D.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19/20, 629–679 (1994)
36. Semeraro, G., Esposito, F., Malerba, D., Fanizzi, N., Ferilli, S.: A logic framework for the incremental inductive synthesis of datalog theories. In: Fuchs, N.E. (ed.) LOPSTR 1997. LNCS, vol. 1463, pp. 300–321. Springer, Heidelberg (1998)
37. Michalski, R.S.: Inferential Theory of Learning. Developing foundations for Multistrategy Learning. In: Michalski, R., Tecuci, G. (eds.) Machine Learning. A Multistrategy Approach, vol. IV, pp. 3–61. Morgan Kaufmann, San Francisco (1994)
38. Zucker, J.D.: Semantic abstraction for concept representation and learning. In: Proceedings of the 4th International Workshop on Multistrategy Learning (MSL), pp. 157–164 (1998)
39. Papadias, D., Theodoridis, Y.: Spatial relations, minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science 11, 111–138 (1997)
40. Egenhofer, M.J.: Reasoning about binary topological relations. In: Günther, O., Schek, H.-J. (eds.) SSD 1991. LNCS, vol. 525, pp. 143–160. Springer, Heidelberg (1991)