Semisupervised Wrapper Choice and Generation for Print-Oriented Documents

Alberto Bartoli, Giorgio Davanzo, Eric Medvet, and Enrico Sorio

Abstract—Information extraction from printed documents is still a crucial problem in many interorganizational workflows. Solutions for other application domains, e.g., the web, do not fit this peculiar scenario well, as printed documents do not carry any explicit structural or syntactical description. Moreover, printed documents usually lack any explicit indication about their source. We present a system, which we call PATO, for extracting predefined items from printed documents in a dynamic multi-source scenario. PATO selects the source-specific wrapper required by each document, determines whether no suitable wrapper exists and generates one when necessary. PATO assumes that the need for new source-specific wrappers is part of normal system operation: new wrappers are generated on-line based on a few point-and-click operations performed by a human operator on a GUI. The role of operators is an integral part of the design and PATO may be configured to accommodate a broad range of automation levels. We show that PATO exhibits very good performance on a challenging dataset composed of more than 600 printed documents drawn from three different application domains: invoices, datasheets of electronic components, patents. We also perform an extensive analysis of the crucial trade-off between accuracy and automation level.

Index Terms—document management, administrative data processing, business process automation, retrieval models, human-computer interaction, data entry

1 INTRODUCTION

Despite the huge advances and widespread diffusion of Information and Communication Technology, manual data entry is still an essential ingredient of many interorganizational workflows. In many practical cases, the glue between different organizations is typically provided by human operators who extract the desired information from printed documents and insert that information in another document or application. As a motivating example, consider an invoice processing workflow: each firm generates invoices with its own firm-specific template and it is up to the receiver to find the desired items on each invoice, e.g., invoice number, date, total, VAT amount. Automating workflows of this kind would involve template-specific extraction rules—i.e., wrappers—along with the ability to: (i) select the specific wrapper to be used for each document being processed (wrapper choice), (ii) figure out whether no suitable wrapper exists, and (iii) generate new wrappers when necessary (wrapper generation). The latter operation should be done promptly and possibly with only one document with a given template as it may not be known if and when further documents with that template will indeed arrive. Existing approaches to information extraction do not satisfy these requirements completely, as clarified below in more detail.

In this paper we propose the design, implementation and experimental evaluation of a system with all these features. Our system, which we call PATO, extracts predefined items from printed documents, i.e., either files obtained by scanning physical paper sheets, or files generated by a computer program and ready to be sent to a printer. PATO assumes that the appearance of new templates is not a sort of exceptional event but is part of normal operation.

Wrapper generation has received considerable attention by the research community in the recent years, in particular in the context of information extraction from web sources [1]–[7]. Wrapper-based approaches fit this scenario very well as they exploit the syntactic structure of HTML documents. In this work we focus instead on printed documents, which are intrinsically different from web pages for two main reasons. First, printed documents do not embed any syntactical structure: they consist of a flat set of blocks that have only textual and geometrical features—e.g., position on the page, block width and height, text content, and so on. Second, the representation of a document obtained from a paper sheet usually includes some noise, both in geometrical and textual features, due to sheet misalignment, OCR conversion errors, staples, stamps and so on. PATO addresses wrapper generation based on a maximum likelihood method applied to textual and geometrical properties of the information items to be extracted [8]. The method is semisupervised in that, when no suitable wrapper for a document exists, PATO shows the document to an operator which then selects the items to be extracted with point-and-click GUI selections.

There are significant differences between web information extraction and our scenario even in the
wrapper choice component. Web information extraction often focuses on a single source at once [5], in which case wrapper choice is not an issue. Recently, the research focus shifted on a multi-source scenario motivated by the interest in integrating deep web sites, i.e., databases which are only accessible by filling search forms on specialized web sites [3], [4], [9]. In those cases, however, it is the system that actively accesses each web site: since the system knows which source is being accessed, the system also knows exactly which wrapper to choose. Our wrapper choice problem is different since our system passively receives documents without any explicit indication of the corresponding source. PATO, thus, is required to infer the source from the document itself. PATO addresses wrapper choice based on a combination of two classifiers applied to image-level properties of the document being processed [10]. One of the classifiers determines whether the document has been emitted by an unknown source, in which case the wrapper generation component comes into play. The other classifier determines the existing wrapper to use.

Our work complements earlier proposals for multi-source wrapper choice and generation by systematically introducing the human-in-the-loop factor, an element that we believe is essential for coping with a dynamic set of sources in a practical setting. PATO accommodates a broad range of automation levels in terms of operator-provided feedback, which may occur independently in the two phases of wrapper choice and wrapper generation. Operators need not have any specific IT-related skills, because their feedback merely consists of basic point-and-click selections on a dedicated GUI to either confirm or reject the system suggestions. These design choices allow tailoring the system to widely differing scenarios easily, including the scenario where 100% extraction accuracy is required. This case corresponds to the maximal amount of operator involvement, because PATO expects a feedback on each processed document. Even in this case, PATO may still provide a practical advantage with respect to traditional (i.e., spreadsheet-based) data entry, since it can greatly reduce the operators’ engagement time without affecting accuracy (see Section 6).

We evaluated the performance of PATO on a challenging dataset, composed of 641 digitized copies of real-world printed documents concerning three different extraction scenarios: (i) date, total, VAT etc. from invoices issued by 43 different firms; (ii) title, inventor, applicant etc. from patents issued by 10 different patent sources; (iii) model, type, weight etc. from datasheets of electronic components produced by 10 different manufacturers. We examined the accuracy of PATO from several points of views and in all the 9 different configurations that it supports, corresponding to all possible combinations of full, partial or no automation at the wrapper choice and generation stages. We also evaluated the time required by human operators for each of the processing steps in which they may be involved, as well as the frequency of their intervention as a function of the configuration options. To place these results in perspective, we estimated the time required by human operators to extract the same information from a printed copy and then fill a spreadsheet. All these data enabled us to gain important insights into the practical impact of the various design options available and the accuracy levels that can be obtained.

2 RELATED WORK

Our work addresses a specific multi-source information extraction problem. In this section we place our contribution in perspective about the existing literature. We discuss approaches designed for web documents separately from those for printed documents.

2.1 Multi-source web information extraction

Multi-source web information extraction is primarily concerned with web sources that are accessed through input elements, typically for performing queries [11]. A crucial step for wrapper generation consists of determining the interface of each source, e.g., how to identify the relevant form inputs in the web page. The position of these elements may widely vary across different sources and there are no textual labels for form inputs that are universally meaningful. Most of the solutions proposed in [11] are based on heuristics that leverage syntactic features of the web documents for identifying candidate input elements. Broadly speaking, our scenario faces similar problems, as a searched information item (e.g., invoice number) can be placed at very different positions among different sources, usually with different textual labels (Num., N., #, Ref., ...) or with no textual label at all. On the other hand, as pointed out in the introduction, our approach to wrapper generation must work at a different abstraction level because printed documents have only textual and geometrical features (which may also be noisy as a result of a scanning process) and do not have any explicit syntactic structure.

Similar remarks apply to all the proposals reviewed in this section, the only exception being the method in [5] that uses only visual features of the rendered web documents. In the cited work a result page produced by a web source is segmented in blocks based on the visual appearance of the page. The blocks are organized in a tree structure and matched to trees for the same source already available and previously annotated with the items to be extracted. The matching uses several heuristics based on hierarchical, textual and geometrical properties of blocks. Our approach works on an unstructured sequence of OCR-generated blocks, each associated with position, size and textual content. The block containing a given information
item is the one which maximizes the probability distributions of block variables, whose parameters are fitted with values obtained from previously annotated documents. It shall also be pointed out that in web information extraction there are often multiple records in each page, i.e., sets of information items following a predefined schema—e.g., a page returned by an e-commerce site contains many results, each composed of name, price, description and so on. It follows that (i) wrappers must be able to identify a varying number of records, and (ii) even a single page may provide opportunities useful for wrapper generation, for example by identifying portions of the page with similar structure or visual appearance—as done in [5].

In our scenario, in contrast, there is only one record for each document: a wrapper must identify exactly one record and the presence of recurring patterns in a page is generally irrelevant for wrapper generation. Wrapper generation for search engine results pages is considered in [1]. The cited work aims at connecting thousands of different search engines and argues that it is not practical to manually generate a wrapper for each source. The approach proposes an unsupervised method for generating a wrapper automatically, which is based on stimulating the source with a sample query and then analyzing the obtained result page. In our scenario we cannot stimulate sources to emit sample documents. Moreover, we need to generate a wrapper as soon as the first document from a new source arrives—which also introduces the problem of realizing that the document has been indeed emitted by a new source.

Search engines are also the topic of [2]. This work proposes a tool which integrates several e-commerce search engines (ESE) making them accessible from a single interface. A dedicated component crawls the web for identifying ESE and clusters them according to the domain—ESE selling similar goods will be placed in the same cluster. The wrapper for each ESE is then generated automatically. A similar problem, which consists of crawling and then clustering hidden-web sources, is tackled in [12], with more emphasis on the ability to cope with a dynamic set of sources.

When multiple sources may be stimulated to provide exactly the same data with different templates, the information redundancy of those sources may be leveraged to improve the corresponding wrappers, as proposed in [3]. While this approach may be often exploited in the web, it cannot be applied in our domain—we cannot stimulate two firms to emit invoices with the very same items. Another approach to wrapper generation is proposed in [4], which combines extraction results obtained with existing wrappers (generated for other sources) with operator-provided domain knowledge. In principle a similar approach could be applied to our case, although it would be necessary to carefully examine the resulting amount of operator involvement and, most importantly, the corresponding IT skills required.

In the recent years several studies on web information extraction have started to focus on more automatic techniques for wrapper generation, which implies considering the role of human operators [1], [4], [12]. A new performance measure called revision aimed at quantifying the amount of human effort has been proposed in [5]. Revision measures the percentage of sources for which the wrapper cannot perform a perfect extraction, the rationale being that a manual correction should suffice to achieve perfect extraction. We also provide special emphasis on human operators but we prefer to quantify the effort involved in using our system as it is: we perform an extensive experimental evaluation of the trade-off between extraction accuracy and time amount of operators’ involvement, including a comparison against a traditional data entry solution, i.e., spreadsheet-based.

The ability to accommodate a large and dynamic set of sources efficiently is a crucial requirement in Octopus [9]. The cited work removes a common assumption in web information extraction, i.e., that the relevant sources for a given domain have been identified a priori, and assumes instead that the operator cannot spend much time on each new data source. Our standpoint is very similar in this respect because PATO assumes that the appearance of new sources is part of normal operation. Leaving the different document representations and internal algorithms aside, PATO is similar to Octopus in that both systems work automatically but allow the operator to provide feedback and correct errors.

A radically different multi-source web extraction problem is considered in [13]. In this case the target consists of automatically extracting tables from web lists, which is difficult because list delimiters are inconsistent across different sources and cannot be relied upon to split lines into fields. The authors use a language model and a source- and domain-independent table model built from a precompiled corpus of HTML tables aimed at identifying likely fields and good alignments: different field value candidates depending on different line splits are evaluated basing on their likelihood according to the language and table models. The approach here presented consists of training a graphical model—conditional random fields (CRF)—with a corpus in which each text line has been previously labeled. Documents are represented in terms of textual features that turn out to be both meaningful and useful for text-only table documents—e.g., number of space indents, all space lines and so on.

For a wider and deeper survey of web information extraction systems please see [6] and [7].
2.2 Information extraction from printed documents

Research efforts on information extraction from printed documents and from web documents have proceeded quite independently from each other. Although the two application domains share many similarities and requirements, the different nature of the documents involved calls for different solutions. Indeed, the two fields tend to use different terminology even for similar concepts. In order to make it easier to appreciate our contribution, in this paper we chose to use the terminology of web information extraction even in this review of the research.

Systems for information extraction from printed documents can be subdivided in two categories, depending on whether all documents are processed with the same wrapper, or each document is processed by a wrapper tailored to the document source. Systems that use the same wrapper for all documents are, broadly speaking, suitable for a single application domain and depend on a fair amount of a priori knowledge about that specific domain—i.e., invoices, medical receipts, and so on [14]–[16]. Moreover, these wrappers are usually limited to documents written in a single language. For example, a precompiled table of “main primary tags” is used in [14] to identify labels of interesting information. Text-based and geometrical information is used in [15] for identifying the desired “tags” to be extracted (the cited work proposes a system that may also be configured to use a wrapper tailored to each source, as discussed below). A system focused on table forms is proposed in [16], where the document is subdivided in boxes and then the boxes containing the values to be extracted are identified based on semantic and geometric knowledge.

Systems that instead have a wrapper for each source have wider applicability but require a preliminary wrapper choice operation [15], [17]–[19]. This operation may be performed in two radically different scenarios: static multi-source scenario, where all sources are known in advance and each document submitted to the system has been certainly emitted by one of these sources; dynamic multi-source scenario, where the set of sources is not known in advance: a document may be associated with one of the sources already known to the system, but it may also be associated with a source never seen before. In the latter case the system must be able to detect the novelty and generate a new wrapper accordingly. Needless to say, the dynamic multi-source scenario is much more challenging but encompasses a much broader range of practical problems. To the best of our knowledge, our work is the first description of a system for information extraction from printed documents in a dynamic multi-source scenario.

Insights about wrapper choice in this context can be found in [20]. This problem is usually cast as a classification problem, each class being composed by documents emitted by the same source. Different document features (e.g., image-level or text features) can be used for the purpose of classifying printed documents [21]–[24], which is a premise for the actual information extraction (that is not considered in the cited works).

A static multi-source approach based only on visual similarity is proposed in [21]. Each document is represented as a Gaussian mixture distribution of background, text and saliency. A nearest-neighbor classifier identifies the document class based on an approximation of the Hellinger distance.

The dynamic multi-source approach proposed in [22] is perhaps the closest to ours. The cited work uses a nearest-neighbor classifier based on the main graphic element of documents emitted by the same source, usually the logo. We use instead a two-stage approach based on the full document image: a nearest-neighbor classifier for detecting whether the document has been emitted by a new source, followed by a multi-class Support Vector Machine which determines the existing source [10] (the reason why we have chosen to use two classifiers is discussed later).

An approach based on image-level features similar to ours, but restricted to a static multi-source scenario, is presented in [23]. A radically different classification approach based on text features (graphs of words) is proposed in [24]. This approach accommodates dynamic multi-source scenarios only in part: the system may detect that a document has not emitted by one of the sources already known, but it is not able to define a new class. Thus, following documents from the same source will be again deemed to belong to an unknown source.

Concerning wrapper generation, most of the commercial solutions currently available require a specialized operator to “draw” a wrapper for the given class of documents. This operation may be performed through a specialized GUI, but it requires specific skills hardly found in data-entry operators, e.g., writing macros, using data description languages or notations, testing the correctness of the description. An essential aspect of our work is that we strive to keep the wrapper generation as simple and lightweight as possible, on the grounds that this operation may be frequent in the scenarios of our interest. With our approach it suffices to point and click on those OCR-generated blocks that contain the desired information. It seems reasonable to claim that any administrative operator without any specific IT-related skill is in position to perform this operation easily.

The approach proposed in [15] is based on a supervised labeling procedure which produces a table (file) of logical objects of interest and related tags, which have been manually located on a number of sample invoices of the given class. Our wrappers determine relevant OCR-generated blocks basing on a
probabilistic approach applied to their geometric and textual properties—size, position, page, text length, text alignment, content. The result is that even when the OCR fails to detect correctly such pieces of text as "Total" or "Price", our system generally is still able to identify the relevant information. On the other side, OCR errors simply prevented the system in [15] from identifying the searched tags or labels.

Semisupervised wrapper generation for digitally-born PDF documents has been proposed in [25]–[27]. Wrapper choice is not addressed, while wrapper generation is based on structural information. The approach proposed in [25] exploits the knowledge represented in an ontology. The document is first subdivided in portions based on heuristics that analyze several visual features, including lines and space distribution. Based on an ontology expressed into an ontology description language, document portions are then translated into sets of rules that constitute a logic program, which can be used for the actual information extraction from the document. In [26] the operator specifies textual and syntactical properties of the items to be extracted. Spatial constraints which link each textual item with its surrounding items (e.g., labels) are then defined using fuzzy logic. The blocks that best satisfy those constraints are finally identified with a hierarchical bottom-up exploration. In [27], the system generates an attributed relational graph describing adjacency relations between blocks. An operator annotates this graph with geometric, logical, structural and content-related block attributes, including the specification of which graph nodes contain data to be extracted.

We explored similar ideas in our early experiments and associated each block with some information about its surrounding blocks. First, by requiring that each block selection by the operator be accompanied by the selection of the corresponding label block (e.g., when selecting a block containing a price item, also the block containing “Amount” or “Total” should be selected). Then, by introducing further block variables measuring the distance from the closest blocks. In either case we could not find any significant improvement, hence we chose to pursue the approach that appeared to be simpler to analyze and implement.

A system for information extraction from forms, and only from them, is proposed in [19]. The system assumes a static multi-source scenario and performs wrapper choice based on a classifier fed with a mixture of textual, graphical and geometrical document features. Wrappers identify the desired information items by means of heuristics based on block position, searched keywords, syntax of block contents. Wrapper generation is not automatic.

Finally, this work integrates our previously separated solutions for wrapper choice [10] and wrapper generation [8]. With respect to the cited works, we show how to include novel document sources in the system seamlessly and without interrupting the normal processing flow; we provide a more compact version of the wrapper construction algorithm; and we assess the role of human operators experimentally, by exploring the trade-off between accuracy of extraction and automation level extensively.

3 Our Framework

Documents of our interest are multi-page images. A document is associated with a schema and a template, as follows. The schema describes the information to be extracted from the document and consists of a set of typed elements: for each element, the document contains zero or one value. For example, a schema could be date, totalAmount, documentNumber; a document with this schema could contain the values "7/2/2011", "23, 79" and no value for the respective elements. Without loss of generality, we consider that the system receives in input a set of documents with the same schema, e.g., invoices, or patents, or electronic datasheets.

The template describes how the information is physically arranged on the pages of a document; a document source generates documents with the same template. The wrapper is the set of rules for locating and extracting the information described by the schema from a document generated by a given source. We assume a multi-source scenario and do not require that the set of sources be known in advance, i.e., we assume a dynamic multi-source scenario.

PATO processes a document \(d\) according to the following workflow.

1) In the wrapper choice stage (Section 4), the system selects the appropriate wrapper \(W\) for \(d\) or, when such a wrapper is not available, prepares itself to generate and use a new wrapper in the next stage.

2) In the blocks location stage (Section 5), the system executes an OCR procedure on \(d\) and then uses \(W\) to locate the required information. That is, for each element \(e\) of the schema, the system locates the rectangular region of \(d\) where the value \(v^*_e\) is graphically represented—we call that region the block \(b^*_e\).

3) Finally, for each element \(e\) of the schema, the system extracts the value \(v^*_e\) from the textual content of the corresponding block \(b^*_e\). This stage consists of applying an element-specific regular expression that aims to remove from the textual content of the block the extra-text that is not part of \(v^*_e\). The details of this stage are largely orthogonal to this work and will not be discussed any further: a method for performing this task is proposed in [28].

The workflow processing stages share access to a knowledge repository containing the data needed for system operation—i.e., information about different
document sources (Section 4), wrappers (Section 5.3) and mapping between them (Section 4).

We carefully designed our system to allow selecting different trade-offs between accuracy and amount of human involvement. As described in the next sections, the two main workflow stages can be configured independently of each other so as to be fully automated, or partly automated, or not automated at all.

## 4 Wrapper Choice

The wrapper choice workflow consists of 4 stages: image processing, feature extraction, novelty detection and classification. The image processing stage includes binarization, deskew and rolling, i.e., an operation aimed at aligning all documents in the same way, by removing the empty part at the top of the document. We found that this operation is often necessary in practice, because images obtained by scanning very similar real-world documents could be significantly different due to human errors made during their digitization—positioning errors on the scanner area, non-standard document sizes, cut documents, and so on. To implement rolling, we identify the upper relevant pixel of the image using an edge recognition algorithm applied to a low-resolution version of the image obtained by resizing the original with a $\frac{1}{4}$ scaling factor; we reduce the image in order to remove the noise caused by the scanner and small texts. We consider the first edge as the upper relevant pixel. To maintain the image size, we remove all the content between the upper relevant pixel and the top border to append it at the end of page.

The feature extraction stage transforms the image into a numerical vector $f$ including: (i) density of black pixels; and (ii) density of the image edges. In detail, we divide the image in a $16 \times 16$ grid and for each cell of the grid we compute the black pixel density, i.e., a number in the range $[0, 100]$ representing the percentage of black pixels in the cell. Then, we reduce the resolution of the original image and apply an edge detector. We repeat the previous procedure on the resulting image consisting only of the edges, i.e., we compute the black pixel density for each cell in a $16 \times 16$ grid. We concatenate the two resulting vectors in order to obtain a features vector $f$ of length $16 \times 16 \times 2 = 512$. For further details please refer to [10].

The next workflow stages are based on the notion of class. Documents with similar graphical appearance belong to the same class. Classes are stored in the knowledge repository as feature vectors sets, as follows. For each class $C$, we store a set $F_C$ composed of up to $k$ documents of that class. If there are less than $k$ documents for a class, $F_C$ contains all the documents available, otherwise it contains the last $k$ documents of the class submitted to the system. After early experiments performed on a small portion of our dataset, we chose $k = 10$. For each class $C$, the system also maintains: (i) the centroid of the elements in $F_C$, denoted $x_C$; (ii) the distances between each element in $F_C$ and $x_C$; (iii) mean $\mu_C$ and standard deviation $\sigma_C$ of these distances. These values are updated whenever the composition of $F_C$ changes.

A predefined knowledge of a few classes is needed in order to initialize the system for the wrapper choice stage: after early experiments performed on a small portion of our dataset, we set this starting knowledge to 7 classes with 2 documents each.

It is important to emphasize that a class description does not provide any clue as to how to locate and extract the desired information from documents of that class. The rules for this task are encoded into wrappers. The association between classes and wrappers is stored in the knowledge repository.

The novelty detection stage is a crucial component for coping with the dynamic multi-source scenario: it determines whether the document can be processed with a wrapper already known to the system, or whether a new wrapper has to be generated (i.e., $W = \emptyset$). Note that in the former case the wrapper is not identified and the choice of the (existing) wrapper to be used will be made in the following classification stage, discussed below. The novelty detection stage works as follows:

1. find the class $C$ with the closest centroid $x_C$ to the features vector $f$ of the input document $d$; let $\delta_C$ denote the distance between $f$ and $x_C$;
2. compute two values $r_C = \mu_C + \alpha \cdot \sigma_C$ and $r_C^* = \epsilon \cdot r_C$ ($\alpha > 0$ and $\epsilon > 1$ are two system parameters);
3. if $r_C < \delta_C$, set $W = \emptyset$ and skip the classification stage, otherwise make the document proceed to the classification stage.

In other words, a new wrapper is needed when the distance of the document $d$ from the closest centroid $x_C$ exceeds a certain threshold. This threshold $r_C^*$ depends on the distances from $x_C$ of the documents that are already known to be instances of class $C$—the closer these documents to $x_C$ the smaller $r_C^*$ and vice versa. The threshold also depends in a similar way on the standard deviation of these distances—the more uniform the distances, the smaller the threshold.

The classification stage is entered only when a new wrapper is not needed. This stage takes the document $d$ as input and outputs the wrapper $W$ to be used for $d$. More in detail, this stage associates $d$ with a class $C$ and the wrapper $W$ associated with the class is then extracted from the knowledge repository (the reason why we do not associate $d$ with the class selected by the novelty detector and use an additional classifier is described below). The classification is performed by a multi-class Support Vector Machine (SVM) with linear kernel. The SVM operates on a feature vector $f'$ obtained from $f$ with a dimensionality reduction based on Principal Component Analysis (PCA). The set of features in $f'$ has been selected so as to ensure
systems works as follows: C and exactly one wrapper.

Since the novelty detector is essentially a k-nearest neighbor (k-NN) classifier, we could have avoided the use of the additional SVM-based classifier. That is, when the novelty detector finds that $\delta_C \leq r_C^t$ for a document $d$, the wrapper $W$ for $d$ could have been set to the class $C$ of the nearest centroid. Our early experiments, however, showed that an SVM-based classifier exhibits much better accuracy than a k-NN classifier.

We attempted to remove the need of a separate novelty detector by using only the SVM classifier, in particular, by taking the probability estimates produced by the classifier as indications useful for novelty detection (as suggested in [29]), but this approach did not yield good results. An option that we did not explore consists of using an additional one-class SVM for novelty detection (as opposed to the multi-class one which we use for classification).

Another design option which we did not explore consists of performing wrapper choice based on the OCR-generated document representation, rather than working exclusively on image-level document features.

4.1 Human Intervention and Updates to the Knowledge Repository

The wrapper choice stage may be configured to work in three different ways based on the value of a parameter $H_{WC}$, which can be one among Unsupervised (wrapper choice never delegated to a human operator), Semisupervised, Supervised (always delegated to a human operator). We remark that the human operator only deals with the notion of graphical similarity—he sees neither classes nor wrappers, human operator only deals with the notion of graphically similar images, which exist only within the system. The mapping between the two notions occurs internally to the knowledge repository, as each class is associated with exactly one wrapper.

In detail, let $d$ denote the document being processed and $C$ the class selected by the SVM classifier. The systems works as follows:

- $H_{WC} = \text{Unsupervised} \implies$ The system operates as in Unsupervised, except that in cases 2 and 3 it requires human intervention, as follows.
  
  The system determines the sequence of four classes, say $C_1, C_2, C_3, C_4$, whose centroids are closest to the feature vector $f$ of $d$ (these classes are thus selected by the novelty detector). Then, the system presents on a GUI an image of $d$ and the image of the most recent document processed by the system for each of $C_1, C_2, C_3, C_4$. We chose to show 4 classes because we found, during our early experiments, that this number suffices to always identify the correct class; the operator may browse through the full list of existing classes, however. The operator is required to choose from two options:
  
  1) Select the document most graphically similar to $d$. In this case the system performs the following (let $C_1$ be the class of the document selected by the operator): (a) $r_C < \delta_C \leq r_C^t \implies$ associates $d$ with $C_1$; (b) otherwise $\implies$ defines a new class $C'$ containing only $d$ and using the (existing) wrapper associated with $C_1$.  
  
  2) Specify that in the list of presented documents there is no one similar enough to $d$. In this case the system defines a new class $C'$ containing only $d$ and associated with a new empty wrapper.

- $H_{WC} = \text{Supervised} \implies$ The system operates as in Semisupervised, except that in case 3 (i.e., $\delta_C \leq r_C$) the human intervention is as follows. The system presents on a GUI a set of images selected as above and augmented with the image of a document associated with the class $C$ selected by the SVM classifier. Class $C$ is indicated as the class suggested by the system for $d$. The operator is required to choose from the same two options as above plus a third option:
  
  1) Select the document most graphically similar to $d$.  
  
  2) Specify that none of the presented documents is similar enough to $d$.  
  
  3) Confirm the suggestion by the system. Options (1) and (2) are handled as above, whereas option (3) results in the association of $d$ with the class $C$ chosen by the classifier. Whenever the human operator modifies the choice suggested by the system, moreover, the system recomputes again its choice for all queued documents and does so before presenting them to the human operator (end of Section 3).

Whenever the wrapper choice stage is completed, the system performs the following tasks:

  1) Records the association between the input document $d$ and the corresponding class $C$. This step also involves following operations: (i) the features vector $f$ of $d$ is added to $F_C$; (ii) if the number of elements of $F_C$ is greater than a predefined
value $k$ ($k = 10$ in our prototype), then the oldest element is removed; (iii) the parameters of $C$ required by the novelty detector are recomputed (centroid, distances: Section 4).

2) Recomputes the PCA parameters, i.e., evaluate again the set of features which grants a proportion of variance greater than 95%.

3) Retrains the SVM classifier.

5 BLOCKS LOCATION

5.1 Overview

The blocks location stage takes as input the document $d$ and the corresponding wrapper $W$ selected by the wrapper choice stage. In case $W$ is empty, $W$ has to be generated by the system with the feedback provided by a human operator. A key strength of our proposal is that the wrapper can be generated automatically from intuitive operations performed on a GUI by an administrative operator without any specific IT-related skills. The operator has merely to point and click on those OCR-generated blocks that contain the desired information (Section 5.4).

The blocks location stage begins with an OCR process, that transforms the document $d$ into a set of blocks, each specified by position, size and content. The position is specified by the page number $p$ and the coordinates $x$ and $y$ from the page origin (upper left corner). The size is specified by width $w$ and height $h$. The content is specified by the single-line textual content $l$, expressed as a character sequence. It follows that a block is a tuple of block attributes $b = \langle p, x, y, w, h, l \rangle$.

The blocks containing the required information are located based on a probabilistic approach that we developed earlier in a more general form [8] and summarized in the next sections in a slightly more compact way, specialized for the specific application domain of interest in this work. The basic idea is as follows. We derived a general form for the probability that a block $b$ contains a value for a given schema element $e$. This function, which we call the matching probability, is a parametric function of the attributes of $b$. The wrapper generation consists of estimating these parameters based on the maximum likelihood approach applied to a set of sample documents. Applying a wrapper to a document $d$ consists of selecting, for each element, the block of $d$ that maximizes the corresponding matching probability.

5.2 Matching probability

The matching probability $P_e$ of a given block is the probability that the block contains a value for the schema element $e$. To simplify the notation, in the following we shall focus on a single schema element and omit the specification of $e$. We want to express this probability, which concerns rather complex events, as a function of simple univariate probability distributions of independent random variables obtained from the block attributes—$p, x, y, w, h, l$.

The matching probability includes 6 attributes whose corresponding 6 random variables are, in general, dependent on each other:

$$P(b) = P(\langle p, x, y, w, h, l \rangle)$$

We identified a domain knowledge-based set of dependencies, which allowed us to elaborate and simplify the form of $P$, as follows.

First, we can use marginalization in order to write $P$ basing on the possible values for the page $p$:

$$P(b) = \sum_k P(b \cap p = k)$$

where $P(b \cap p = k)$ is the joint probability of the following two events: $b$ is the matching block and $p = k$. These two events are in general dependent. For example, consider a template of invoices where the total amount value may be located at the bottom of the first page or at the top of the second page: it follows that small values for $y$ are more probable if the page attribute is equal to 2 and large values for $y$ are more probable if the page attribute is equal to 1—in other words, the $y$ attribute of the block is dependent on the $p$ attribute. Accordingly, we can rewrite the joint probability in terms of conditional probability on the page $p$:

$$P(b \cap p = k) = P(b|p = k)P(p = k) = P_k(\langle x, y, w, h, l \rangle)P(p = k)$$

where $P_k(\langle x, y, w, h, l \rangle)$ is the probability that a block identified by the tuple $\langle x, y, w, h, l \rangle$ is the matching block, given that its page attribute is equal to $k$. Concerning $P(p = k)$, we assume that there is a finite set $K = \{k_1, k_2, \ldots\}$ of possible values for $p$, whose corresponding probabilities are $s_{k_1}, s_{k_2}, \ldots$. In other words, $P(p = k) = s_k I(p; k)$, where $I(p; k) = 1$ for $p = k$ and 0 otherwise and $s_k = 0$ if $k \notin K$. In summary, we may rewrite $P$ as follows:

$$P(b) = \sum_k s_k I(p; k)P_k(\langle x, y, w, h, l \rangle)$$

Concerning $P_k(\langle x, y, w, h, l \rangle)$, we assume that $y$ and $h$ are independent from the other three variables. In particular, note that, since blocks contain exactly one line of text, the height attribute $h$ is largely independent from its text content $l$. Hence, we can write:

$$P_k(\langle x, y, w, h, l \rangle) = P^y_k(y)P^h_k(h)P^xw_l(\langle x, w, l \rangle) \quad (1)$$

Concerning now $P^xw_l(\langle x, w, l \rangle)$, we split the $x, w, l$ dependency in one between $x$ and $w$ and another between $w$ and the text content $l$.

- The dependency between $x$ and $w$ represents the fact that a given text could be aligned in three different ways: left, center or right (justified text
may be handled in any of these three cases, for the purpose of this analysis). It follows that:

- in case of left-alignment, \( x \) and \( w \) are independent;
- in case of center-alignment, \( x^c = x + \frac{w}{2} \) and \( w \) are independent;
- in case of right-alignment, \( x^r = x + w \) and \( w \) are independent.

The dependency between \( w \) and \( l \) represents the fact that, in general, the longer the text content, the larger the block width. We define \( w' = \frac{w}{L(l)} \) as the average width of the characters composing the block text content, being \( L(l) \) the number of characters in \( l \); we assume that \( w' \) and \( l \) are largely independent, since \( w' \) depends on the font size and type, rather than on the text content.

We can hence write \( P^x.w.l((x, w, l)) \) in three possible forms depending on text alignment:

\[
P^x.w.l((x, w, l)) = \begin{cases} 
P^x_k(x) P^w_k(w') P^l_k(l), & \text{left} \\
P^x_k(x^c) P^w_k(w') P^l_k(l), & \text{center} \\
P^x_k(x^r) P^w_k(w') P^l_k(l), & \text{right} 
\end{cases}
\]

which can be summarized as:

\[
P^x.w.l((x, w, l)) = P^x_k(x') P^w_k(w') P^l_k(l)
\]

where \( x' \) is a shortcut symbol which represents one among \( x^c \) and \( x^r \).

Finally, we obtain the following general form for the matching probability:

\[
P(b) = \sum_k s_k I(p; k) P^y_k(y) P^h_k(h) P^x_k(x') P^w_k(w') P^l_k(l)
\]

Note that each \( P_k \) is a univariate distribution.

At this point we have to choose a form for each of the above distributions. We assume that the size attributes \( (w' \) and \( h) \) and the position attributes \( (x' \) and \( y \) can be described as random variables with Normal Distribution (denoted by \( N(\mu, \sigma) \)). In a preliminary phase of our study, we considered using other distributions for the aforementioned random variables—in particular the Uniform Distribution and the Kernel Density Estimation (Parzen windows)—but we found the Normal Distribution models them better.

Concerning the textual content, \( P^l_k(l) \) is the probability that the text \( l \) is the text of the matching block; \( P^l_k \) hence operates on text, differently from all other probabilities which operate on numbers. We assume that \( P^l_k(l) \) can be expressed as a Markov chain of order 2. Its state space corresponds to the set of possible characters and 2 pseudo-characters representing the begin and end of the text. The probability \( M \) of the text \( l \) is defined as the probability of the state sequence corresponding to \( l \). For example, the probability of the word "tw o" is given by \( P(\text{"t"} \blacktriangleright \text{"w"} \blacktriangleright \text{"o"})P(\text{"w"} \blacktriangleright \text{"o"})P(\text{"o"} \blacktriangleleft \text{"w"}) \), where \( \blacktriangleright \) and \( \blacktriangleleft \)

represent the begin and the end of the text, respectively, and each transition probability corresponds to an element of the transition matrix \( T_k \). The details about how we set \( T_k \) are given in Section 5.3. The final form for the matching probability is the following:

\[
P = \sum_k s_k I(k) N(\mu^y_k, \sigma^y_k) N(\mu^h_k, \sigma^h_k) N(\mu^x_k, \sigma^x_k) N(\mu^w_k, \sigma^w_k) N(\mu^l_k, \sigma^l_k) \mathcal{M}(T_k)
\]

where we omitted the function arguments for readability.

The wrapper, thus, merely requires the set of values for the parameters of Equation 2 (one set of parameters for each schema element): \( s_k, \mu^y_k, \sigma^y_k, \mu^h_k, \sigma^h_k, \mu^x_k, \sigma^x_k, \mu^w_k, \sigma^w_k, \mu^l_k, \sigma^l_k \) and \( T_k \). In the next section we describe how we generate the wrapper, i.e., how we choose these values.

We analyzed the impact of the various parameters on performance. The key result of the analysis, omitted for brevity, is that removing any one feature rendered the wrapper too inaccurate to be useful. In particular, text and position are a powerful combination of features but they are not enough.

5.3 Wrapper generation

A wrapper requires a set of values for the parameters described in the previous section. These values are selected based on the maximum likelihood method with respect to the distributions assumed in Equation 2. The maximum likelihood is computed based on the ground truth information stored in the knowledge repository for a few documents. The ground truth for a wrapper \( W \) and an element \( e \) takes the form of a set of true blocks, denoted \( B_e \). These values are initialized and updated as described in this section and the following one.

The procedure for generating a wrapper is as follows. We will describe the procedure for one single element, since the matching probability for an element can be generated independently from the matching probabilities for the other elements.

- We set \( s_k \) to the frequency of the true blocks in \( B \) whose page \( p \) is \( k \).
- For each \( k \)-th Normal Distribution of Equation 2, we estimate its \( \mu_k \) and \( \sigma_k \) parameters, using the corresponding attributes of the true blocks in \( B \) whose page \( p \) is \( k \). In order to prevent overfitting, we impose a lower bound for the \( \sigma \) parameter, which we denote with \( \sigma_{xy} \) for the position attributes and \( \sigma_{w,h} \) for the size attributes; \( \sigma \) is also set to this lower bound in case \( B \) contains only one element.
- For each \( k \), we choose the \( x, w, l \) dependency which maximizes the probability of blocks in \( B \) whose page \( p \) is \( k \).
After early experiments performed on a small portion of our dataset, we set our parameters as follows: \( \sigma_{xy} = \sigma_{wh} = 0.05 \) inches = 1.27 mm, \( \epsilon = \frac{1}{2} \).

Finally, concerning the text probability \( \mathcal{M}_k \), we perform some preprocessing on each piece of text \( l \): (i) transform to lowercase, (ii) replace all digit characters with "#", (iii) replace all space characters with standard space character, (iv) replace all punctuation characters with "." and finally (v) replace any other character with ",". We denote the result with \( l' \). Recall that we can describe a chain of order 2 using a transition matrix \( T \) of size \( a \times a^2 \), being \( a \) the number of states. In our case, given the aforementioned text elaborations, \( a = 32 \) and \( T \) indexes represent respectively a single character and a sequence of two characters: e.g., \( l_{3,34} = t_{\text{"bc"}, \text{"ab"}} = P(\text{"bc"} | \text{"ab"}) \).

In order to set the \( T_k \) matrix for \( \mathcal{M}_k \) we start with a frequency matrix \( F \) with the same size of \( T_k \) and each element set to 0. Then, we process the textual content \( l' \) of each true block \( B \) whose page \( p \) is \( k \) and increment the corresponding \( F \) elements. For example, after processing the sequence ",banana" we will have \( f_{\text{"a"}, \text{"b"}} = f_{\text{\"ba"}, \text{\"ab"}} = f_{\text{\"a"}, \text{\"n"}} = 1 \) and \( f_{\text{\"n"}, \text{\"a"}} = 2 \). At the end, we set for each pair of indexes \( u, v \):

\[
t_{u,v} = \begin{cases} 
(1 - \epsilon) \frac{f_{u,v}}{\sum_{z=1}^{a^2} f_{u,z}}, & \text{if } f_{u,v} > 0 \\
\epsilon \frac{f_{u,v}}{N_u}, & \text{otherwise} 
\end{cases}
\tag{3}
\]

where \( N_u \) is the number of \( f_{u,v} \) which are greater than 0. We use the term \( \epsilon \) to make the text probability smoother, i.e., so that it assigns non-zero (yet low) probabilities also to textual contents which are not present in the training set.

## 5.4 Human Intervention and Updates to the Knowledge Repository

The block selection stage may be configured to work in three different ways based on the value of a parameter \( H_{BL} \), which can be one among Unsupervised, Semisupervised, and Supervised. Each value provokes a different behavior in terms of human intervention and determines whether wrappers are updated during the normal system operations, as described in full detail below. The value for \( H_{BL} \) can be selected independently of the value selected for the analogous parameter \( H_{WC} \) in the wrapper choice stage.

Wrappers are stored in the knowledge repository in the form of: for each element \( e \), (i) a set of true blocks \( B_e \), and (ii) values for the parameters of the matching probability. The parameters of the matching probability are computed by using \( B_e \), as explained in the previous section. These parameters are recomputed whenever the composition of a \( B_e \) changes.

The set \( B_e \) contains up to \( w \) elements, where \( w \) is a system-wide parameter (we chose \( w = 20 \) according to the experimental findings of [8]). If there are less than \( w \) elements, parameters are computed based on all the true blocks available, otherwise one could choose the \( w \) true blocks of the schema element in many different ways. Our prototype uses the last \( w \) true blocks found by the system.

Operations in the block selection stage depend on whether the input wrapper \( W \) to use is empty or not. If it is not, processing proceeds as follows:

- **\( H_{BL} = \text{Unsupervised} \) \implies \text{Human intervention is never required.}** For each element \( e \) of the schema, the matching block \( b_e \) is added to the set of true blocks \( B_e \). If the set of true blocks \( B_e \) contains more than \( w \) elements, the oldest one is removed.

- **\( H_{BL} = \text{Semisupervised} \) \implies \text{Human intervention is required only for those elements whose set of true blocks } B_e \text{ contains } 4 \text{ elements or less.}** In this case, the system presents to the operator an image of the document \( d \) with the matching block \( b_e \) highlighted and offers two options:
  1. Confirm the choice made by the system.
  2. Select a different block.

The GUI has been carefully designed in the attempt of speeding up the processing of each document as much as possible. In particular, the system highlights all the matching blocks, i.e., one block for each schema elements. If all the choices are correct, the processing of the entire document requires one single click on the “confirm” option. Otherwise, the operator corrects only the wrong blocks and then clicks on “confirm”.

The knowledge repository is then updated as in the Unsupervised case, with the true blocks either selected or confirmed on the GUI.

If the set of true blocks \( B_e \) contains more than 20 elements, the oldest one is removed.

- **\( H_{BL} = \text{Supervised} \) \implies \text{Human intervention is always required.}** In this case, the system presents each document to the operator and processing proceeds as in \( H_{BL} = \text{Semisupervised} \).

If the input wrapper \( W \) is empty, human intervention is always required irrespective of the value of \( H_{BL} \). The reason is because, for each element, the set of true blocks is empty and hence the parameters have no values.

In this case the GUI presents a document with no highlighted block and offers only the option of selecting the true block (i.e., there is no true block to confirm). Once the document has been processed by the operator, the selected true blocks are inserted into the respective sets. The processing of further documents associated with this wrapper will now proceed as above, based on the value of \( H_{BL} \). In other words, the processing of the first document of a wrapper requires a human operator that points and clicks on the relevant information, hence allowing the system to generate the wrapper. The processing of further documents of that wrapper may or may not require a
human operator depending on the configured value for $H_{BL}$.

6 EXPERIMENTS AND RESULTS

6.1 Prototype

We implemented PATO prototype according to the SaaS (software as a service) approach. Documents and results can be submitted and received, respectively, in batches with a programmatic REST interface. Operators interact with PATO through a browser-based GUI. We developed the back-end with Java Enterprise Edition 6 and used Glassfish as application server. We used the Google Web Toolkit framework (GWT) for the user interface and the open-source OCR Cuneiform 1.1.0.

All the experiments described in the next sections were executed on a quad core 2.33 GHz PC with 8 GB of RAM. The average time needed for choosing the wrapper for a document is 1.4 ms. The update of knowledge repository portion related to the wrapper choice stage takes, on the average, 2.1 s; most of this time is spent for retraining the SVM classifier. Regarding the block location stage, the average time for generating or updating a wrapper is 31.5 ms. The average time for locating an element on a document is about 135 ms. To place these figures in perspective, it suffices to note that the OCR execution of a single-page document takes about 4 s, whereas image preprocessing (binarization, deskew) takes 2.4 s.

6.2 Dataset

A common problem in the research field of information extraction from printed documents is the lack of a reference testing dataset, which makes it impossible to compare results obtained by different research groups. Following common practice, thus, we built three datasets:

- **invoices**: we collected a corpus of 415 real invoices issued by 43 different firms, i.e., with 43 different templates. The distribution of documents amongst firms is not uniform, the largest number of documents from the same firm being 80. This dataset is challenging because all documents were digitized after being handled by a corporate environment, thus they are quite noisy as they contain stamps, handwritten signatures, ink annotations, staples and so on. We defined a schema composed of 9 elements: date, invoiceNumber, total, taxableAmount, vat, rate, customer, customerVatNumber, issuerVatNumber.

- **patents**: we collected 118 patents from 8 different countries which use different templates. We defined a schema composed of 10 elements: classificationCode, filingDate, title, abstract, representative, inventor, publicationDate, applicationNumber, priority, applicant.

- **datasheets**: we collected 108 diode datasheets, produced by 10 different companies which use different templates. We defined a schema composed of 6 elements: model, type, case, storageTemperature, weight, thermalResistance.

The invoices and a small portion of the other datasets were acquired at 300 DPI, while the remaining documents were born-digital. We manually constructed the ground truth for all the documents. An operator visually inspected each document and, for each element of the schema, manually selected the corresponding true block, if present. We made part of our dataset publicly available.

6.3 Experiments

The nature of the problem implies that the dataset has to be handled as a sequence rather than as a set: the results may depend on the specific order in which the documents are submitted to the system.

Based on these considerations, we performed 20 executions of the following procedure for each dataset:

1. we constructed a random sequence $S$ with documents of the dataset;
2. we submitted $S$ to the system;
3. we repeated the previous step 9 times, one for each possible combination of values for $H_{WC}$ and $H_{BL}$. We hence performed 540 experiments.

As stated in previous sections, we selected values for the system-wide and wrapper-independent parameters after a few exploratory experiments on a small portion of the first dataset and left these values unchanged in all the experiments presented here.

In order to explore the trade-off between accuracy and automation level, we measured several quantitative indexes, as follows. We measured accuracy after the two salient workflow stages:

1) **wrapper choice (WC) accuracy**: the fraction of documents which are associated with the correct existing wrapper or that correctly lead to the generation of a new wrapper;
2) **block location (BL) accuracy**: the fraction of found blocks which correctly match the corresponding true blocks.

The block accuracy is a cumulative measure, i.e., it takes into account the effect of any possible error in the prior wrapper choice stage.

We estimated the automation level by counting, in each of the 540 experiments, the number of times that the system asked input from human operators. In particular, we counted the number of confirmations and the number of selections provided by the operators, i.e., the number of correct and of wrong suggestions, respectively. Input from human operators was actually simulated based on the ground truth.

1. 1 http://machinelearning.inginf.units.it/data-and-tools/
ghega-dataset (invoices are not included due to privacy concerns)
We also cast the previous automation level results in terms of time required by human operators. To this end, we executed a further dedicated experiment for assessing the time required by each basic GUI operation, as follows. We selected a panel of 7 operators and a random sequence of 32 documents, one sequence for each dataset, the same sequence for all operators. We instrumented the system to measure the elapsed time for the processing steps of the operator and configured the system with $H_{WC} = H_{BL} = \text{Supervised}$. Then, we asked each operator to process the sequence and collected the corresponding measurements. The results are 5.24 s and 5.80 s for the WC confirmation and selection respectively, and 3.45 s and 5.08 s for the BL confirmation and selection respectively.

In order to obtain a baseline for these values and for the system as a whole, we also made a rough assessment of the time required by a traditional data entry procedure. For each dataset, we prepared a spreadsheet with the same schema as in the previous experiments and printed 10 documents selected at random from the dataset. Then, we asked each of the 7 operators to fill the spreadsheets starting from the printed documents. The average time for document turned out to be 89.23 s for the invoices documents, 50.9 s for the datasheets and 114.7 s for the patents. As will be shown in the next section, these values are significantly higher than the average time for document required by PATO, even in the configuration with maximal amount of operator involvement (i.e., $H_{WC} = H_{BL} = \text{Supervised}$). We did not even attempt to translate the resulting estimates of time savings into monetary values because of the excessive number of factors that would be necessary for obtaining useful figures: labour cost, country, terms of service regarding accuracy and latency, pipeline structuring (e.g., web-based crowdsourcing or actual shipping of printed documents) and so on. Indirect but meaningful indications in this respect may be found on more specialized studies. In [31] the processing cost of a printed invoice has been estimated in about $13 per document and [18] reports that the number of printed paper forms which are handled yearly by Japanese public administration is greater than 2 billions. The economic impact of accountant personnel workload involved in generating and maintaining invoice wrappers for a widely adopted invoice recognition software is quantified in [32].

### 6.4 Results

Accuracy results averaged on the three datasets are shown in Table 1, columns WC accuracy and BL accuracy. The table contains one row for each combination of the system parameters $H_{WC}$ and $H_{BL}$. The main finding is that block location accuracy is greater than 71% even in the most automated configuration. The experiments also demonstrate that less automated configurations indeed improve accuracy, as expected. Most importantly, even a moderate amount of operators’ feedback in the block selection stage allows obtaining a block location accuracy consistently above 89%—it suffices to exclude the $H_{BL} = \text{Unsupervised}$ mode; please recall that $H_{BL} = \text{Semisupervised}$ involves the operator only when there are less than 5 examples for an element. It is worth pointing out also that the 100% accuracy in block location with $H_{BL} = \text{Supervised}$ could not be taken for granted in advance, as this accuracy index takes into account any possible error in the previous wrapper choice stage. Indeed, the block location stage turns out to be quite robust with respect to errors in the wrapper choice stage: despite different accuracy levels of the wrapper choice stage, the block location accuracy is always quite high. It can also be seen, from the standard deviation of the accuracy (Table 1), that the system is robust with respect to the order in which documents are submitted.

These results cannot be compared directly to other earlier works, due to the lack of a common benchmark dataset (see previous section). However, it is interesting to note that, even in the most automated configuration, our numerical values are at least as good as those reported in earlier works focused on a static multi-source scenario for printed documents, e.g., [15], [33], even though we are coping with a much more challenging dynamic scenario.

Automation level results are also shown in Table 1, columns WC interaction and BL interaction. These columns show the average number of interactions required for each document, in each configuration. For example, with $H_{WC} = \text{Unsupervised}$ and $H_{BL} = \text{Unsupervised}$, the operator is required to perform for each document, on the average, 0.67 block selections, no block or wrapper confirmation, no wrapper selection.

These figures are cast in terms of average human interaction time per document in the last column of Table 1, based on our measurements for WC/BL selection and confirmation (see previous section). Interestingly, the human processing time in supervised modes (25.71–30.49 s) is significantly smaller than 86.9 s, i.e., our estimate for the document processing time without our system (see previous section). In other words, our system may provide a practical advantage also in those scenario where perfect accuracy is required, since it can reduce the human processing time by 66% with respect to traditional data entry.

Data in Table 1 are averaged on the three datasets. Table 2 shows accuracy and time length of human interaction for each document, along with the traditional data entry processing time (baseline), separately for each dataset.

We remark again that in the field of information extraction from printed documents there does not exist any standard dataset benchmark. Our indexes,
Table 1

Average accuracy with standard deviation, number and time length of human interactions per document for both wrapper choice and block location stages. Each row of the five rightmost columns corresponds to a different combination for the automation level parameters $H_{WC}$ and $H_{BL}$. Results are averaged on the three datasets.

Table 2

Results for the three datasets with two combinations of $H_{WC}$ and $H_{BL}$ parameters.

7 CONCLUDING REMARKS

We have presented the design, implementation and experimental evaluation of a system (PATO) for information extraction from printed documents in a dynamic multi-source scenario. PATO assumes by design that the appearance of new sources is not a sort of exceptional event and is able to generate corresponding new wrappers: wrapper generation is supervised by the human operator, who interacts with the system quickly and simply, without the need of any dedicated IT skills. A crucial consequence of this assumption is that a wrapper must be generated using only information that can be extracted with a simple point-and-click GUI. Wrappers are indeed generated based on a maximum-likelihood approach applied on geometrical and textual properties of documents. PATO chooses the appropriate wrapper for a document based on a combination of two classifiers (k-NN and SVM) that operate on image-level features. The prototype allows offering the service on a cloud-based platform and may support crowdsourcing-based interaction modes.

We assessed the performance of PATO on a challenging dataset composed of more than 640 printed documents. The result are very satisfactory and suggest that our proposal may indeed constitute a viable approach to information extraction from printed documents. The system provides very good accuracy even in the most automated configuration. On the other hand, PATO can deliver perfect block location accuracy (i.e., 100% with our dataset) and, at the same time, reduce significantly the operators’ engagement time with respect to traditional data entry.

Acknowledgments

This work has been partly supported by MIDA4. The authors are grateful to Luca Bressan and Marco Mauri for their help in the implementation, and to the anonymous reviewers for their detailed and insightful comments.

REFERENCES


