# Visual Information Extraction [*]

Yonatan Aumann [†‡] Ronen Feldman [†‡§] Yair Liberzon [‡]
Benjamin Rosenfeld [‡] Jonathan Schler [†]

† Department of Computer Science, Bar Ilan University, Ramat Gan 52900, Israel
{aumann,feldman}@cs.biu.ac.il
‡ ClearForest Ltd., 6 Yoni Netanyahu St., Or Yehuda 60376, Israel

**Abstract**

Typographic and visual information is an integral part of textual documents. Most information extraction systems ignore most of this visual information, processing the text as a linear sequence of words. Thus, much valuable information is lost. In this paper, we show how to make use of this visual information for information extraction. We present an algorithm that allows to automatically extract specific fields of the document (such as the title, author, etc.), based exclusively on the visual formatting of the document, without any reference to the semantic content. The algorithm employs a machine learning approach, whereby the system is first provided with a set of training documents in which the target fields are manually tagged, and automatically learns how to extract these fields in future documents. We implemented the algorithm in a system for automatic analysis of documents in PDF format. We present experimental results of applying the system on a set of financial documents, extracting nine different target fields. Overall, the system achieved a 90% accuracy.

**Keywords.** Information Extraction, PDF Analysis, Text Analysis, Wrapper Induction

## 1 Introduction

Most information extraction systems [1, 2, 3, 4, 5] simplify the structure of the documents they process by ignoring much of the visual characteristics of the document, e.g. font type, size and location, and process the text as a linear sequence. This allows the algorithms to focus on the semantic aspects of the document. However, valuable information is lost. Consider, for example, an article in a scientific journal. The title is readily recognized based on its special font and location, but less so based on its semantic content, which may be similar to the section headings. Similarly, for the author names, section headings, running title, etc. Thus, much important information is provided by the visual layout of the document. In this paper we present an information extraction system that is based solely on the visual characteristics of the document, and show that this visual information alone is sufficient to provide high accuracy extraction, for specific fields (e.g. the title, author names, publication date, etc.). We note that the visual approach provided by this paper is

[*]An preliminary description of this work appeared in [28]
[§]Corresponding author.

not aimed at replacing the semantic one, but rather on complementing it. It can also function as a preprocessor or a converter to other systems.

We provide a general algorithm which allows to perform the IE task based on the visual layout of the document. The algorithm employs a machine learning approach whereby the system is first provided with a set of training documents in which the desired fields are manually tagged. Based on these training examples the system automatically learns how to find the corresponding fields in future documents.

**Problem Formulation.** A document $D$ is a set of *primitive elements* $D = \{e_1, \ldots, e_n\}$. A primitive element can be a character, a line, or any other visual object, depending on the document format. A primitive element may have any number of visual attributes, such as font size and type, physical location, etc. The *bounding box* attribute, which provides the size and location of the bounding box of the element, is assumed to be available for all primitive elements. We define an *object* in the document to be any set of primitive elements.

The *Visual Information Extraction (VIE) task* is as follows. We are provided with a set of *target fields* $F = \{f_1, \ldots, f_k\}$, to be extracted, and a set of *training documents* $\mathcal{T} = \{T_1, \ldots, T_m\}$ wherein all occurrences of the target fields are tagged. Specifically, for each target field $f$ and training document $T$, we are provided with the object $f(T)$ of $T$ that is of type $f$ ($f(T) = \emptyset$ if $f$ does not appear in $T$). The goal is that when presented with an un-tagged query document $Q$, to correctly tag the occurrences of the target fields that exist in $Q$ (not all target fields need be present in each document).

**Results and Paper Organization.** We present a general framework and algorithm for the VIE task. We show that the VIE task can be decomposed into two subtasks. First, for each document (both training and query) we must group the primitive elements into meaningful objects (e.g. lines, paragraphs, etc.), and establish the hierarchical structure among these objects. The details of this stage are explained in Section 2. Then, in the second stage, the structure of the query document is compared with those of the training documents to find the objects corresponding to the target fields. This stage is detailed in Section 3.

We also show how to improve the results by introducing the notion of *templates* which are groups of training documents with a similar layout (say, articles from the same journal). Using templates we can identify the essential features of the page layout, ignoring particularities of any specific document. Templates are detailed in Section 3.2.

We implemented the system for a VIE task on a set of documents containing financial analyst reports. The documents were in PDF format. Target fields included the title, authors, publication dates, and others. The implementation and the results are detailed in Section 4.

## 1.1 Related Work

Information extraction (IE) from written text is an area of much research and significant results. Information extraction is a subfield of natural language processing that is concerned with identifying predefined types of information from text. Much of the work on information extraction was driven by the Message Understanding Conferences (MUC) [1, 2, 3, 4, 5], which provided a uniform framework for the evaluation and comparison of different approaches and systems. There are multiple different tasks within the general task of IE. In MUC-7, for example, five tasks have been defined:

1. Named Entity (NE) Extraction: Tagging of specific entities mentioned in the text, e.g. person, organization, geographic location, etc.

2. Co-reference Resolution: Identification of distinct noun-phases within the text that refer to the same entity.

3. Attribute Assignments (also called Template Elements (TE)): Determining the different attributes of a given entity. For an entity of type *person*, for example, attributes include: "descriptor" (captain, Dr., etc.), and "category" (military, academic, etc.).

4. Relations Extraction (also called Template Relations (TR)): Determining the relations between elements, as indicated by the text (e.g. an "Employment" relation, consisting of slots for *employer*, *employee*, and *position*).

5. Scenario Extraction: Extraction of scenarios, which are events, possibly described over multiple text locations.

Traditionally, IE systems, including all those in the MUC evaluations, base the extraction on the content of the text itself. In fact, the MUC data was provided in raw ASCII format, with essentially no physical layout information. In this paper, in contrast, we base the extraction solely on the physical layout of the text.

Extraction from structured and semi-structured text is frequently performed using *wrappers* [22], most commonly for extraction from world-wide-web sources. A *wrapper* is a procedure for the extraction of specific information from a specific data source. The output of the wrapper is provided in a predefined structured format (typically a database schema). Wrappers are commonly used by software agents [15, 17, 29]. A wrapper consists of a set of rules that determine how to identify the relevant information, based on the structure of the source.

Developing tools to facilitate wrapper generation has been the subject of much research (see [13]). To aid with manual creation of wrappers special tool-kits have been developed [19, 26]. Graphical interfaces for semi-automatic wrapper generation allow for interactive visual definition of wrappers (see [8, 10]). Machine learning techniques have been employed extensively for the automatic generation of wrappers (also called *wrapper induction*), using a multitude of techniques (see [21, 20, 24]). Several of the algorithms allow the handling of both semi-structured and unstructured text [30, 11, 16].

The system described in this paper can also be viewed as a type of automatic wrapper generator. However, the system is fundamentally different from traditional wrappers. Traditional wrappers operate on the textual *representation* of the document, e.g. the HTML document, and base the extraction on clues provided by this representation, e.g. the HTML tags. Our visual extraction system, in contrast, bases its extraction on the actual visual appearance of the document (as generated by the textual representation). To see the difference, consider, for example, a specific physical layout of a document. This physical layout can be generated in multiple different ways in PDF (two column text, for example, can be provided column by column or line by line). Using traditional wrappers, these two representation would necessitate two different wrappers. Our visual extraction system, in contrast, operates on both representation identically, since they both represent the same visual layout.

Several works consider analysis of PDF documents. The AIDAS system [7] is designed to automatically index technical manuals provided in PDF form. The system determines the logical structure of the document (sections, item lists, titles, etc.) using a bottom-up process combined with the use of top-down (shallow) grammars. Lovegrove and Brailsford [23] describe a system for analysis of PDF documents based on the blackboard method. Futrelle et al. [18] describe a system for analysis of diagrams in PDF documents, using support vector machines. Chao, Beretta and Sang [12] describe a system for the analysis of PDF documents aimed at reuse of its logical elements.

Logical structure analysis and document understanding for documents available in raster format has been studied extensively. Most of these systems are based on rules (or grammars) that represent apriori knowledge on the overall logical structure of document. In this case the main challenge is to correlate between this apriori structure and the raster information. Several systems for raster document analysis use machine earning techniques. The system developed as part of the WISDOM project is of special interest in this regard (see [14, 6, 9]). The WISDOM++ system accepts a scanned document as input and transform it into rich XML format. The processing of the document in WISDOM++ involves several steps. The *document understanding* step is analogous to the task we consider here. The WISODOM++ system employs a machine learning approach to document understanding. This is achieved by automatically learning a set of rules, using first-order learning systems. The descriptions of both layout structures and models are given in a suitably defined first-order language, with functions expressing unary properties, such as height and length, and binary predicates and function expressing interrelationships among layout components (e.g., contain, on-top, and so on). The rules are learned from the training data and applied to the test. Our system, in contrast, does not operate by learning the specific rules pertaining to the individual objects, but rather by attempting to match the overall structure of the training examples to that of the test document.

Learning techniques have also been used extensively in the general theory of computer vision, and for visual object recognition in particular (see [27, 25]). The problem we consider here is on the one hand more restricted, as we consider only textual documents, but on the other hand also more demanding, as the differences between the fields may be subtle.

## 2 Structural Layout Analysis

Recall that a document is a set of primitive elements, such as characters, figures, etc. The *objects* of a document are sets of primitive elements. Target fields, in general, are objects. Thus, the first step in the Visual IE task is to group the primitive elements of the documents into higher-level objects. The grouping should provide the conceptually meaningful objects of the document, such as paragraphs, headings, footnotes, etc. For humans, the grouping process is easy, and is generally performed unconsciously based on the visual structure of the document. The goal is thus to mimic the human perceptual grouping process. This process is often called *structural layout analysis.*

## 2.1 Problem Formulation

We model the structure of the objects of a document as a tree, where leaves are primitive elements and internal nodes are composite objects. We call this structure the O-Tree (Object-Tree) of the document. The O-Tree structure creates a hierarchal structure among objects, where higher-level objects consist of groups of lower-level objects. This hierarchal structure reflects the conceptual structure of the document, where objects such as columns are groups of paragraph, which, in turn, are groups of lines, etc. The exact levels and objects represented in the O-Tree are application and format dependent. For an HTML document, for example, the O-Tree may include objects representing tables, menus, text body, etc., while for a PDF documents the O-Tree may include objects representing paragraphs, columns, lines, etc. Accordingly, for each file format and application we define the *Type Hierarchy*, $\mathcal{H}$, which determines the set of possible *object types*, and a hierarchy among these objects. Any Type Hierarchy must contain the type DOCUMENT, which must be at the root of the hierarchy. When constructing an O-Tree for document, each object is labelled by one of the *object types* defined in the Type Hierarchy, and the tree structure must correspond to the hierarchical structure defined in the hierarchy.

Formally, an Type Hierarchy, $\mathcal{H}$, is a labelled rooted DAG (Directed Acyclic Graph), such that:

- The leaf nodes contain all the possible types for primitive elements.
- Internal nodes contain all the possible types for composite objects.
- The root node is the type DOCUMENT.
- For types $x$ and $y$, type $y$ is a child of $x$ if an object of type $x$ can (directly) contain an object type $y$.

For a document $D = \{e_1 \ldots, e_n\}$ and an Type Hierarchy $\mathcal{H}$, an *O-Tree of $D$* according to $\mathcal{H}$ is a tree, $O$, such that:

- The leafs of $O$ consist of all primitive elements of $D$.
- Internal nodes of $O$ are objects of $D$.
- If $X$ and $X'$ are nodes of $O$ (objects or primitive elements) and $X \subset X'$ then $X'$ is an ancestor (or parent) of $X$.
- Each node $X$ is labelled by a label from $\mathcal{H}$, denoted $\mathit{label}(X)$.
- If $X'$ is a parent of $X$ in $T$ then $\mathit{label}(X')$ is a parent of $\mathit{label}(X)$ in $\mathcal{H}$.
- $\mathit{label}(\mathit{root}) =$ DOCUMENT.

## 2.2 Algorithm

Given a document, we construct an O-Tree for the document. In doing so, we wish to construct objects best reflecting the true grouping of the elements into "meaningful" objects (e.g. paragraphs, columns, etc.). In doing so we rely solely on the *physical* representation of the document and not on any hidden tags that the document may contain (i.e. we do not rely on XML, HTML or PDF tags describing the nature of the object, such as “Header”, “Title”, etc.). When constructing an object we see to it that:

- The elements of the objects are within the same physical area of the page. Specifically, we require that any object n the O-Tree is *connected*, i.e. for an object $X$, it cannot be decomposed into two separate objects $X_1$ and $X_2$, such that any line connecting $X_1$ and $X_2$ necessarily crosses an element in a different object.

- The elements of the object have similar characteristics (e.g. similar font type, similar font size, etc.). Specifically, we assume the existence of *fitness function* $\mathit{fit}(\cdot,\cdot)$, such that for any two objects $X$ and $Y$, where $\mathit{label}(Y)$ is child of $\mathit{label}(X)$, $\mathit{fit}(Y,X)$ provides a measure of how fit is $Y$ as an additional member to $X$ (e.g. if $X$ is a paragraph and $Y$ a line, then how similar is $Y$ to the other lines in $X$). We add $Y$ to $X$ only if $\mathit{fit}(Y,X)$ is above some threshold value, $\gamma$. The exact nature of the function $\mathit{fit}(\cdot,\cdot)$ and the threshold value are format and domain dependent.

Given these two criteria, we construct the O-Tree in a greedy fashion, from the bottom up, layer by layer. In doing so, we always prefer to enlarge existing objects of the layer, starting with the largest object. If no existing object can be enlarged, and there are still “free” objects of the previous layer, a new object is created. The procedure completes when the root object, labelled Document is completed. A description of the algorithm is provided in Figure 1.

## 3 Structural Mapping

Given a Visual Information Extraction task, we first construct an O-Tree for each of the training documents, as well as for the query document, as described in the previous section. Once all the documents have been structured as O-Trees, we need to find the objects of $Q$ (the query document) that correspond to the target fields. We do so by comparing the O-Tree of $Q$, and the objects therein, to those of the training documents. This we perform in two stages. First we find the training document that is (visually) most similar to the query document. Then, we map between the objects of the two documents to discover the targets fields in the query document.

### 3.1 Basic Algorithm

**Document Similarity.** Consider a query document $Q$ and training documents $\mathcal{T} = \{T_1, \ldots, T_n\}$. We seek to find the training document $T_{opt}$ that is *visually* most similar to the query document. We do so by comparing the O-Trees of the documents. In the comparison we only concentrate on similarities between the top levels of O-Tree. The reason is that even similar documents may still differ in the details.

Let $O(Q)$ and $O(T_1), \ldots, O(T_n)$, be the O-trees of the query document and the training documents, respectively, and let $\mathcal{H}$ be the Type Hierarchy. We define a subgraph of $\mathcal{H}$, called the *Signature Hierarchy* and denoted by $\mathcal{S}$, consisting of the types in $\mathcal{H}$ that determine features of the global layout of the page (e.g. columns, tables). The exact types included in the signature are implementation dependent, but, in general, the signature would include the top one or two levels of the Type Hierarchy. For determining the similarity between objects we assume the existence

Input: $D$ - Primitive elements of a document
Output: O-Tree for $D$

```
1   For each type $t \in \mathcal{H}$ do
1.      let $level(t)$ be the length of the longest path from $t$ to a leaf
2.  Let $h = level(\textsc{Document})$
3.  $Objects(0) \leftarrow D$
4.  For $i = 1$ to $h$ do
5.      $Objects(i) \leftarrow \emptyset$
6.      $free \leftarrow Objects(i-1)$
7.      While $free \neq \emptyset$ do
8.          Foreach $X \in Objects(i)$ in order of descending size do[1]
9.              Foreach $Y \in free$ in order of increasing distance from $X$ do[1]
10.                 If $Y$ is a neighbor of $X$ and $fit(Y, X) \geq \gamma$ then
11.                     $X \leftarrow X \cup Y$
12.                     make $Y$ a child of $X$
13.                     Remove $Y$ from $free$
14.                     Break (goto line 7)
15.         Foreach $t \in \mathcal{H}$ such that $level(t) = i$ do
16.             if $Objects(i)$ does not include an empty object of type $t$
17.                 Add an empty set of type $t$ to $Objects(i)$[2]
18.     end while
19.     Remove empty objects from $Objects(i)$
20. end for
21. return resulting O-Tree
```

[1]Break ties randomly.

[2]Empty objects have size zero, and are neighbors, with maximum distance, of all other objects.

Figure 1: Constructing the O-Tree

of a similarity function $sim(X, Y)$, which provides a measure of similarity between objects of the same type, based on the object characteristics such as size, location, fonts, etc. ($sim(X, Y)$ is implementation dependent).

Given a query document $Q$ and a training document $T$, for each object $X$ in the signature of $T$ we find the object $X'$ of $Q$ (of the same type as $X$) that is most similar to $X$. We then compute the average similarity for all objects in the signature of $T$ to obtain an overall similarity score between $Q$ and $T$. We choose the document, $T_{opt}$, with the highest similarity score. A description of the procedure is provided in Figure 2.

```
Input: $Q$, $T_1, \ldots, T_n$, and their respective O-trees
Output: $T_{opt}$ (training document most similar to query document)


1   for $i = 1$ to $n$ do
1.      $total \leftarrow 0$
2.      $count \leftarrow 0$
3.      Foreach $t \in \mathcal{S}$ do
4.          Foreach $X \in O(T_i)$ of type $t$ do
5.              $s(X) \leftarrow \max\{sim(X, X') \mid X' \in O(Q), X'$ of type $t\}$
6.              $total \leftarrow total + s(X)$
7.              $count \leftarrow count + 1$
8.      $score(i) \leftarrow total/count$
9.  end for
10. $opt \leftarrow \text{argmax}\{score(i)\}$
11. return $T_{opt}$
```

Figure 2: Finding the most similar document

**Finding the Target Fields.** Once the most similar training document, $T_{opt}$, has been determined, it remains to find the objects of $Q$ that correspond to the target fields, as tagged in the document $T_{opt}$. We do so by finding, for each target field $f$, the object within the O-Tree of $Q$ that is most similar to $f(T_{opt})$ (the object in $O(T_{opt})$ tagged as $f$). Finding this object is done in an exhaustive manner, going over all objects of $O(Q)$. We also make sure that the similarity between this object and the corresponding object of $T_{opt}$ is above a certain threshold, $\alpha$, or else we decide that the field $f$ has not been found in $Q$ (either because it is not there, or we have failed to find it). A description of the procedure is provided in Figure 3.

Note that the tagging of the training documents is performed prier (and independently) to the construction of the O-trees. Thus, tagged objects need not appear in the O-tree. If this is the case, line 2 sees to it that we take the minimal object of $O(T_{opt})$ that fully contains the tagged object.

```
Input:
• $Q$, $T_{opt}$ (and their respective O-trees)
• $\{f(T_{opt}) \mid f \in F\}$ (target fields in $T_{opt}$)
Output: $\{f(Q) \mid f \in F\}$ (Target fields in $Q$)

1   Foreach $f \in F$ do
1.      Let $\bar{f}(T_{opt}) \in O(T_{opt})$ be minimal such that $f(T_{opt}) \subseteq \bar{f}(T_{opt})$
2.          $f(Q) \leftarrow \text{argmax}\{sim(\bar{f}(T_{opt}), X) \mid X \in O(Q), X \text{ of type } t\}$
3.          if $sim(\bar{f}(T_{opt}), f(Q)) < \alpha$ then $f(Q) \leftarrow \emptyset$
```

Figure 3: Finding the Target Fields

### 3.2 Templates

The above algorithm is based on finding the single most similar document to the query document, and then extracting all the target fields based on this document alone. While this provides good results in most cases, as we shall see in Section 4, there is the danger that particularities of any single document may reduce the effectiveness of the algorithm. To overcome this problem we introduce the notion of *templates*, which allow to compare the query document to a *collection* of similar documents. A *template* is a set of training documents that have the same general visual layout, e.g. articles from the same journal, web pages from the same site, etc. The documents in each template may be different in details, but share the same overall structure.

Using templates, we find the template that is most similar to the query document (rather than the document most similar). We do so by, for each template, averaging the similarity scores between the query document and all documents in the template. We then pick the template with the highest average similarity. Once the most similar template is determined, the target fields are provided by finding the object of $Q$ most similar to a target field in *any* of the documents in the template. A description of this stage is provided in Figure 4.

## 4 Experimental Results

We implemented the system for Visual Information Extraction, as described above, on documents that are analyst reports from several leading investment banks.

**The Data.** The data consisted of a total of 285 analyst reports from leading investment banks: 71 from BearSterns, 29 from CSFB, 26 from Dresdner, and 159 from Morgan Stanley. All documents were in PDF format. The documents were clustered into 30 templates: 7 in the Bear Sterns data, 4 in the CSFB data, 5 in the Dresdner data, and 14 in the Morgan Stanley Data. All documents were manually tagged for the target fields. The target fields included the following fields: AUTHOR, TITLE, SUBTITLE, COMPANY, TICKER, EXCHANGE, DATE, GEOGRAPHY, INDUSTRY INFO. Not all documents included all target fields, but within each template, documents had the same target

```
Input:
• $Q$ and its O-Tree
• $\mathcal{T}_{opt} = \{T_1, \ldots, T_k\}$ (most similar template) and respective O-Trees
• $\{f(T) \mid f \in F, T \in \mathcal{T}_{opt}\}$ (target fields in $\mathcal{T}_{opt}$)
Output: $\{f(Q) \mid f \in F\}$ (Target fields in $Q$)

1   Foreach $f \in F$ do
1.      Foreach $T \in \mathcal{T}_{opt}$ do
2.          Let $\bar{f}(T) \in O(T)$ be minimal such that $f(T) \subseteq \bar{f}(T)$
3.              $X_T \leftarrow \text{argmax}\{sim(\bar{f}(T), X) \mid X \in O(Q), X \text{ of type } t\}$
4.              $s(T) \leftarrow sim(\bar{f}(T), X_T)$
5.      if $\max\{s(T) \mid T \in \mathcal{T}_{opt}\} \geq \alpha$ then
6.          $T_{\max} \leftarrow \text{argmax}\{s(T) \mid T \in \mathcal{T}_{opt}\}$
7.          $f(Q) \leftarrow X_{T_{\max}}$
8.      else $f(Q) \leftarrow \emptyset$
```

Figure 4: Finding the Target Fields with Templates

fields. The dataset is available on line at www.cs.biu.ac.il/∼aumann/datasets/KAIS05.zip .

**Implementation.** The Type Hierarchy ($\mathcal{H}$) used in the system is provided in Figure 5. The Signature Hierarchy contained the objects of type COLUMN and PARAGRAPH. The implementation of the fitness function *fit*$(\cdot,\cdot)$ (for the fitness of one object within the other) takes into account the distance between the objects and the similarity in fonts. For the fitness of a line within an existing paragraph, it also takes into account the distance between lines. The similarity function *sim*$(\cdot,\cdot)$, measuring the similarity between objects in different documents, is based on similarity between the sizes and locations of the respective bounding boxes.

**Results.** Tests were performed using five-fold cross validation. In each template the documents were randomly divided into five subsets. We then performed five tests as follows. In each test, for each template, one of the five subsets served for training and the other four were used for the test. (Of course, all test documents were placed together, without distinction of the source template; it was up to the algorithm to determine the right template.) We measured the performance of the system with the basic algorithm (Section 3.1) and with the use of templates (Section 3.2). The overall average recall and precision values are provided in Figure 6. On the whole, the introduction of templates improved the performance of the algorithm, increasing the average accuracy from 83% to 91%. We note that for both algorithms the recall and precision values were essentially identical. The reason is that for any target field $f$, on the one hand each document contains only one object of type $f$, and, on the other hand, the algorithm marks one object as being of type $f$. Thus, for every recall error there is a corresponding precision error. The slight difference that does exist between the recall and precision is due to the cases where the algorithm decided not to mark any

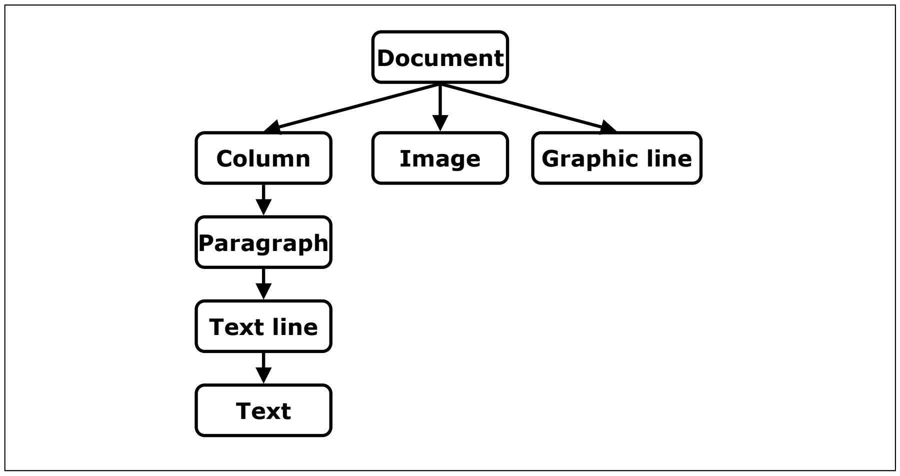

Figure 5: The Type Hierarchy.

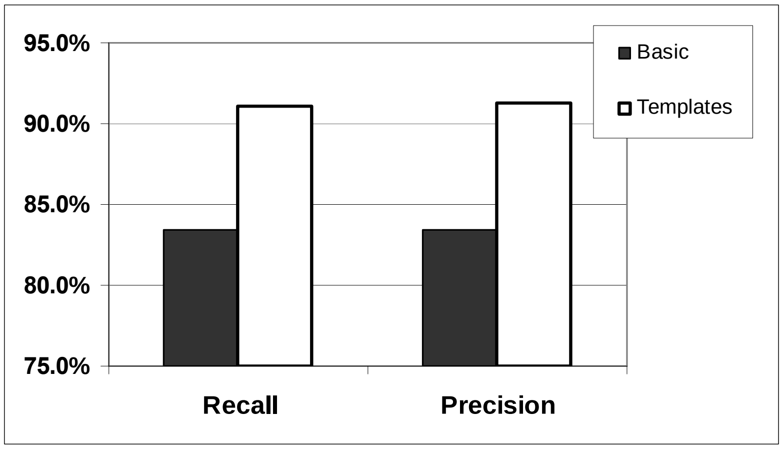

Figure 6: Overall average recall and precision rates for basic algorithm and for algorithm using templates.

element, in which case there is a recall error but not a precision error.

Some target fields are harder to detect than others. Figure 7 provides the accuracy rates for the different target fields. It is interesting to note that while the introduction of templates improves the accuracy in most cases, there are some target fields for which it reduces the accuracy. Understanding the exact reasons for this, and how to overcome such problems, is a topic for further research.

**Performance.** The system was implemented in C++, on a Microsoft Windows platform. We measured the performance of the system using a laptop with a Pentium-4 1.7 GHZ Centrino processor, with 1 GB of RAM. Figure 8 shows the performance of the system, in seconds per test document, as a function of the number of training documents. As can be seen, the time complexity is essentially linear in the number of training documents. This is expected, since the comparison with the test document is performed separately for each training document. Figure 9 depicts the performance of the system, in seconds per test document, as a function of the number of templates. Here too, the time complexity was linear in the number of templates, though with a much flatter

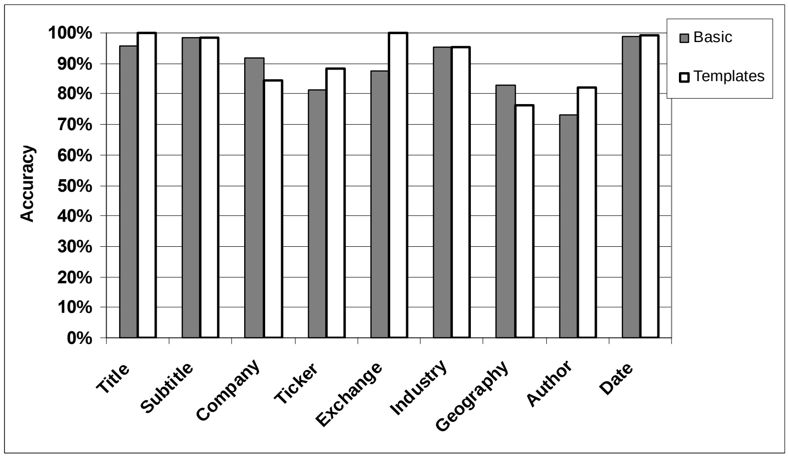

Figure 7: Accuracy (recall and precision) rates for the different target fields.

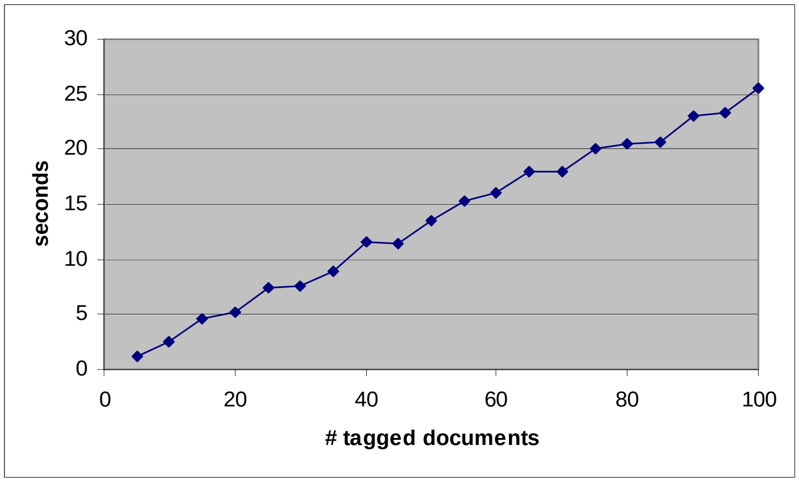

Figure 8: Time complexity as a function of the number of training documents. Time provided for single test document.

curve. This is again expected, as there is an additional overhead for determining the similarity score for each template, but this process is relatively fast. Throughout, the performance was linear in the number of test documents.

## 5 Discussion

Typographic and visual information is an integral part of textual documents. All but the most basic document formats provide for extensive representation of visual information (font sizes and types, indentations, etc.). The importance of the visual formatting is not only to please the eye. Rather, the visual information supplements the textual one, providing visual clues as to the meaning and/or role of the different parts of the document. The clues can be typographic (e.g. italization), location based (e.g. indentation), or any other form of visual representation (e.g. putting an important piece of text within a box). The text itself, without its visual formatting, is much more difficult

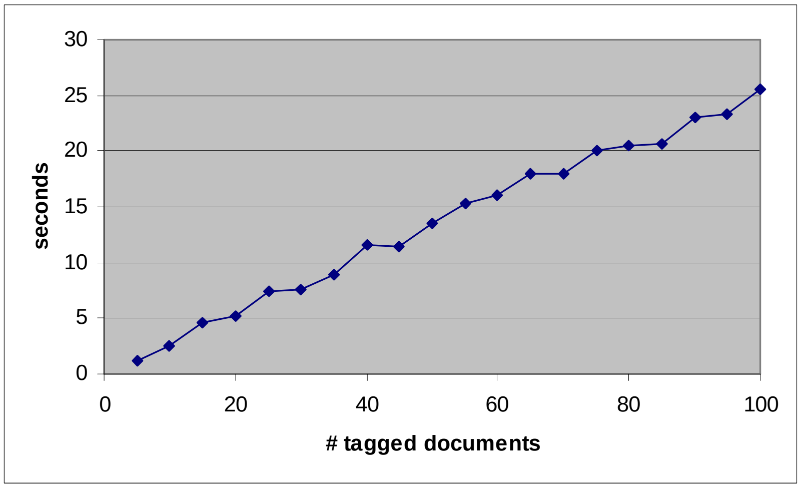

Figure 9: Time complexity as a function of the number of templates. Time provided for single test document.

to understand. Accordingly, we believe that it is important to consider the visual formatting when analyzing textual documents.

In this paper we show how the visual formatting can be efficiently used to extract specific fields, such as the title, subtitle, author names, etc., from textual documents. The algorithm we present employs a machine learning approach, whereby the system is provided with training documents wherein the target fields are manually tagged, and automatically learns how to extract these fields in future documents. We present experimental results in implementing the system for PDF documents, achieving accuracy levels of 90%.

The algorithm and system presented in this paper rely exclusively on the visual appearance, totally ignoring the semantic content. In this respect the approach is opposite to most information extraction systems, which focus only on the sematic content. The results presented here show that the visual information is sufficiently rich to allow for highly accurate extraction of specific fields. Furthermore, we show that the specific visual clues of each target field can be automatically learned by the system.

Clearly, the visual approach presented here also has its limitations. First and foremost, the visual approach can only capture fields with distinct visual characteristics, such as the title, authors, publication date, etc. Semantic elements mentioned within the running text, such as people names, locations, etc., clearly cannot be detected by the visual approach. In addition, the learning process presented here only works for features and structures that have a relatively high level of consistency among documents, such as title, author, etc. The method would be less applicable to structures with a high level of variations between documents.

Ultimately, we believe that a complete solution for information extraction should make use of the entire spectrum of available information: semantic, syntactic and visual. In such a system, the visual approach presented here would be one of the components in a combined, integrated approach. The development of such a system is left for future research. We note that Barardi et al [9] describe an integrated approach based on the WISDOM system.

## References

[1] *Proceedings of the Seventh Message Understanding Conference (MUC-7).* Available at: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_toc.html.

[2] *Proceedings of the Third Message Understanding Conference (MUC-3).* Morgan Kaufmann, 1991.

[3] *Proceedings of the Forth Message Understanding Conference (MUC-4).* Morgan Kaufmann, 1992.

[4] *Proceedings of the Fifth Message Understanding Conference (MUC-5).* Morgan Kaufmann, 1993.

[5] *Proceedings of the Sixth Message Understanding Conference (MUC-6).* Morgan Kaufmann, 1995.

[6] O. Altamura, F. Esposito, and D. Malerba. Transforming paper documents into XML format with WISDOM++. *International Journal of Document Analysis and Recognition*, 4(1):2–17, 2001.

[7] A. Anjewierden. AIDAS: Incremental logical structure discovery in pdf documents. In *Proceedings of the sixth International Conference on Document Analysis and Recognition (ICDAR)*, pages 374–378.

[8] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.

[9] M. Berardi, M. Lapi, and D. Malerba. An integrated approach for automatic semantic structure extraction in document images. In S. Marinai and A. Dengel, editors, *Document Analysis Systems*, volume 3163 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2004.

[10] L. Bright, J.R. Gruser, L. Raschid, and M. E. Vidal. A wrapper generation toolkit to specify and construct wrappers for Web accessible data sources (WebSources). *International Journal of Computer Systems Science and Engineering*, 14(2):83–97, 1999.

[11] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 328–334, 1999.

[12] H. Chao, G. Beretta, and H. Sang. PDF document layout study with page elements and bounding boxes. In *Workshop on Document Layout Interpretation and its Applications (DLIA2001)*, 2001.

[13] L. Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

[14] F. Esposito, D. Malerba, and F.A. Lisi. Machine learning for intelligent processing of printed documents. *Journal of Intelligent Information Systems*, 14(2/3):178–198, 2000.

[15] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, 1994.

[16] D. Freitag. Toward general-purpose learning for information extraction. In *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 404–408, 1998.

[17] M. Friedman and D. S. Weld. Efficiently executing information-gathering plans. In *15th International Joint Conference on Artificial Intelligence*, pages 785–791, Nagoya, Japan, 1997.

[18] R. P. Futrelle, M. Shao, C. Cieslik, and A. E. Grimes. Extraction, layout analysis and classification of diagrams in PDF documents. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 1007–1015. IEEE, 2003.

[19] J. Hammer, H. García-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. In *Proceedings of the Twenty-Third ACM SIGMOD International Conference on Management of Data*, pages 532–535, 1997.

[20] C. N. Hsu and M. T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[21] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[22] J.W. Lewis. Wrappers: integration utilities and services for the DICE architecture. In *Proceedings of the Second National Symposium on Concurrent Engineering*, pages 445–457, 1991.

[23] W. S. Lovegrove and D. F. Brailsford. Document analysis of PDF files: Methods, results and implications. *Electronic Publishing*, 8(2,3):207–220, June/September 1995.

[24] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1-2):93–114, 2001.

[25] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, June 2000.

[26] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. D. Ullman. A query translation scheme for rapid implementation of wrappers. In *4th Intenational Conference on Deductive and Object-Oriented Databases*, volume E1013 of *LNCS*, pages 319–344. Springer, 1995.

[27] T. Poggio and S. Edelman. Network that learns to recognize 3D objects. *Nature*.

[28] B. Rosenfeld, R. Feldman, and Y. Aumann. Structural extraction from visual layout of documents. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 203–210, 2002.

[29] E. Selberg and O. Etzioni. The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert*, 12(1):8–14, 1997.

[30] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.