IJDAR (2001) 4: 2–17

International Journal on Document Analysis and Recognition

Transforming paper documents into XML format with WISDOM++

Oronzo Altamura, Floriana Esposito, Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari, via Orabona 4, 70126 Bari, Italy;
e-mail: {altamura,esposito,malerba}@di.uniba.it

Received June 15, 2000 / Revised November 7, 2000

Abstract. The transformation of scanned paper documents to a form suitable for an Internet browser is a complex process that requires solutions to several problems. The application of an OCR to some parts of the document image is only one of the problems. In fact, the generation of documents in HTML format is easier when the layout structure of a page has been extracted by means of a document analysis process. The adoption of an XML format is even better, since it can facilitate the retrieval of documents in the Web. Nevertheless, an effective transformation of paper documents into this format requires further processing steps, namely document image classification and understanding. WISDOM++ is a document processing system that operates in five steps: *document analysis, document classification, document understanding, text recognition* with an OCR, and *text transformation* into HTML/XML format. The innovative aspects described in the paper are: the preprocessing algorithm, the adaptive page segmentation, the acquisition of block classification rules using techniques from machine learning, the layout analysis based on general layout principles, and a method that uses document layout information for conversion to HTML/XML formats. A benchmarking of the system components implementing these innovative aspects is reported.

Key words: Document image analysis – Layout analysis – Induction of decision trees – Transformation into HTML/XML format

1 Introduction

Recent advances in information and communication technologies have increased the need for tools that are able to transform data presented on paper into a web-accessible format, such as HTML/XML. There are several benefits to this transformation: the HTML/XML version of a document can be accessed via Internet more quickly than the original bitmap, the user can manipulate the original document or search for particular words, and it is possible to define some hypertext structures which improve document reading (Worring and Smeulders 1999). Commercial OCR systems are still far from supporting this function satisfactorily. Most of them can save scanned documents in HTML format, but generally their appearance on the browser is not similar to the original documents. Rendering problems, such as missing graphical components, wrong reading ordering in two-columned papers, missing indentation and broken text lines, are basically due to poor layout information extracted from the scanned document. Moreover, no style sheet is associated to documents saved in HTML format, so the presentation of textual information cannot be customized for viewing.

Another limitation concerns the HTML language itself, which cannot represent the logical document structure. XML (eXtensible Markup Language) is a meta-language, that is, a language that describes other languages, developed by the World Wide Web Consortium (1998). It inherits some characteristics from HTML and describes the content of documents that are stored in electronic format. The most significant feature of XML, optional but powerful, is the concept of *DTD* (*D*ocument *T*ype *D*efinition), which specifies the logical hierarchy of documents and can make information retrieval on the Web easier (e.g., XML-QL is a language designed to express database-style queries in XML documents (Deutsch et al. 1999)). Generally, a DTD is associated to a class of documents with the same logical structure. For instance, the DTD of home pages will define some logical elements, such as name, surname, address, profession, and so on. The reason why commercial OCR systems cannot generate documents in full XML format (that is, with a DTD) is that they do not perform document image understanding (Wang et al. 1999).

In order to transform printed documents into HTML/XML format it is necessary to have knowledge of both layout and logical structures, which are extracted by image analysis and understanding processes. WISDOM++ (http://www.di.uniba.it/∼malerba/wisdom++/) is a doc-

Correspondence to: D. Malerba; e-mail: malerba@di.uniba.it

ument image analysis system that can transform paper documents into either HTML or XML format by operating in five steps: *document analysis, document classification, document understanding, text recognition* with an OCR and *transformation* into a web-accessible format. The document analysis process automatically extracts the hierarchical layout structure used in the transformation into HTML format, while the document classification process automatically recognizes the membership class used to associate a style sheet to a document. In the case of transformation into XML format, the result of the document classification process affects the choice of both the style sheet and the DTD, while the results of the document understanding and text recognition steps define the content of the .xml file. Documents in XML format are a convenient means for storing and exchanging all the data on the geometrical and logical structures extracted in the five processing steps above (Hitz et al. 1999; Simske 1999).

One of the distinguishing features of WISDOM++ is the use of a rule base to support some tasks performed in the first three steps. The rule base is automatically built from a set of training documents using symbolic machine learning tools and techniques, which make the system highly *adaptive* (Esposito et al. 1999). *Real-time* user interaction is another relevant feature that affects the design of the whole system. The behavioral design of the interface partially follows the *sequential interaction style*, in which the user action is controlled by the system itself (Hix and Hartson 1993), and the *glass-box model*, in which some system mechanisms are revealed to the user (Wenger 1988).

In addition to document image generation in the HTML/XML version, WISDOM++ presents several novelties compared with its predecessor, PLRS (Esposito et al. 1994).

1. Some decision tree learning techniques are now applied to the *block classification* problem, that is the separation of text blocks from graphics, while the first release of PLRS used a linear pattern classifier for this task.
2. The symbolic learning technique applied to build the rule base for the document classification and understanding steps has been extended in order to handle both numeric and symbolic data. Indeed, some experimental results on the document understanding problem (Malerba et al. 1997a) have led to the conclusion that mixed numeric/symbolic descriptions are essential for generating accurate models of logical structures.
3. WISDOM++ can manage *multi-page* documents, each of which is a *sequence* of pages. The user is responsible for the definition of the right sequence, since the optical scan function is able to work on a single page at a time. Pages of multi-page documents are processed independently of each other in all steps.
4. WISDOM++ has also been designed as a multi-user system, in the sense that each authorized user has his/her own rule base. Two categories of users are currently defined: *administrators* and *final users*. Administrators are responsible for establishing which classes of documents each final user can manage, for determining the logical components that can be detected in each class of documents, and for training the system with a set of documents.

This paper provides a comprehensive explanation of the document analysis process performed by the new system WISDOM++, whose earlier versions have been partially described in previous works (Malerba et al. 1997b; Esposito et al. 1999; Altamura et al. 1999). In particular, Sect. 2 describes the main preprocessing step, namely deskewing, implemented in the document analysis system, and Sect. 3 illustrates the adaptive document block segmentation and classification techniques. The knowledge-based extraction of the layout structure is covered in Sect. 4, while the adaptive document classification and understanding is briefly reviewed in Sect. 5. Section 6 explains the layout-based transformation of scanned paper documents into a form suitable for an Internet browser. Finally, some experimental results are reported in Sect. 7.

2 Data capture and preprocessing

Each page of a multi-page document is optically scanned with a resolution of 300 dpi and thresholded into a binary image. The bitmap of an A4-sized page takes about $2,496 \times 3,500 = 1,092,000$ bytes and is stored in TIFF format. The image is associated with a coordinate system whose origin is in the top left-hand corner; the X-coordinate increases from the leftmost to the rightmost column, while the Y-coordinate increases from the uppermost to the lowest row.

The segmentation of the page is performed by a top-down method, which is quite fast, but generally ineffective when applied to skewed documents. Consequently, the skew angle has to be estimated and corrected by inverse rotation of the document image. Following Baird's definition (1987), the skew angle of a document image I is the orientation angle θ of its text baselines. It is positive when the image is rotated anti-clockwise, otherwise it is negative. Contrary to some systems that can perform local skew angle estimation (Antonacopoulos 1997; Yu et al. 1995), WISDOM++ can correctly manage only documents with the same skew angle for all text lines. The estimation $\hat{\theta}$ of the actual skew angle θ is obtained as the composition of two functions: $S(I)$, which returns a *sample region* R of the document image I, and $E(R)$, which returns the *estimation* of the skew angle in the sample region R. The selection of a sample region is peculiar to WISDOM++ and has the advantage of reducing the computational cost of the estimation step, while its main disadvantage is the possibility of errors in the estimation of the *dominant* (i.e., the most frequent) skew in documents with many local skews for text lines.

In order to select the sample region WISDOM++ computes both the horizontal projection profile H of the document image and the average number of pixels per row (*avpx*). Then it extracts a set of *regions* from H:

a region R_i is a sequence of adjacent rows in H, whose height is greater than *avpx/4*. In this way, only regions with prominent peaks will be considered, since $E(R_i)$ is more likely to be close to the true skew angle θ. Each region is classified as horizontal line, text, or image as specified in the paper by Altamura et al. (1999). Since the focus is on the estimation of the skew angle of text regions, the system selects, if any, the text region R_i with the maximum average density of black pixels per row. Otherwise, the system returns the region, classified as horizontal line or image, satisfying the following conditions: its base is smaller than 310 pixels and it has the maximum average density of black pixels per row[1].

Once the sample region R has been selected, $E(R)$ is computed. Let H_θ be the horizontal projection profile of R after a virtual rotation of an angle θ. The histogram H_θ shows sharply rising peaks with a base equal to the character height when text lines span horizontally, while it presents smooth slopes and lower peaks when the skew angle is large. This observation is mathematically captured by a real-valued function, $A(\theta) = \sum_{j \in R} H_\theta^2(j)$, which has a global maximum at the correct skew angle. Thus, finding the actual skew angle means locating the global maximum value of $A(\theta)$. Since this measure is not smooth enough for the application of gradient techniques, the system adopts some peak-finding heuristics, which is simpler than that proposed by Baird (1987). Initially, it takes thirty-two samples of A(θ) for rotation steps of 20 pixels[2], thus only thirty-two rotations are possible for steps of 0.46°.. Then it selects the angle θ' maximizing $A(\theta)$ and rotates the sample region at finer steps (10 pixels), starting from $\theta' - 30$, until a peak is found. The skew angle is finally estimated by computing the vertex of the parabola interpolating three points around the peak in the space $(\theta, A(\theta))$ (see Fig. 1). The estimated skew angle is used as a default value when the user asks the system to rotate the document. The user can repeat the loop "skew estimation – document rotation" until satisfying results are obtained.

In the preprocessing phase the *spread factor* of the document image is also computed. It is defined as the ratio of the average distance between the regions R_i *(avdist)* and the average height of the same regions *(avheight)*. In quite simple documents with few sparse regions this ratio is greater than 1.0, while in complex documents with closely written text regions the ratio is lower than the unit. The spread factor is used to define some parameters of the segmentation algorithm. At the end of the preprocessing phase, the resolution of the document image is reduced from 300 to 75 dpi, which is a reasonable trade-off between the accuracy and the speed of the segmentation process. In this way, noisy black specks on a white background are also filtered out.

[1] When no full region satisfying these conditions exists, a sub-region of exactly 310 pixel is selected.

[2] For an A4-sized document image with 2,496 columns, this correspond to rotations of an angle equal to *arctan* (20/2496) $\approx$ 0.46°. WISDOM++ can detect skews smaller than $\pm$7.2°

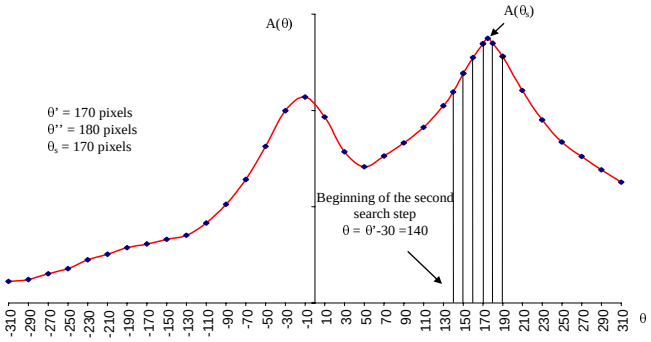


Fig. 1. An example of peak search performed by WISDOM++. The original document image has been rotated by −4°

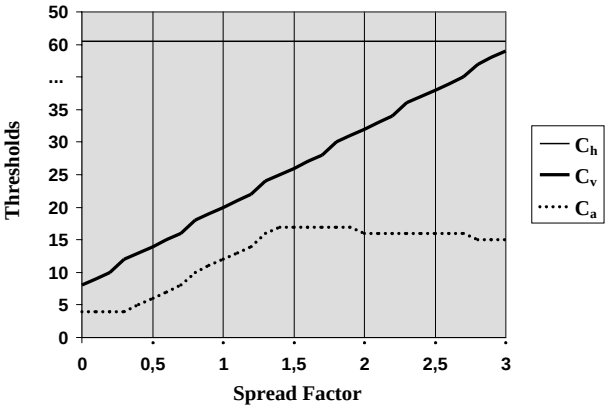


Fig. 2. Adaptive threshold definition depending on the spread factor

The size of the reduced bitmap for an A4-sized page is about 70 kb.

3 Adaptive page segmentation and block classification

WISDOM++ segments the reduced document image into rectangular blocks by means of a variant of the Run Length Smoothing Algorithm (RLSA) (Wong et al. 1982). The RLSA applies four operators to the document image: (1) horizontal smoothing with a threshold C_h; (2) vertical smoothing with a threshold C_v; (3) logical AND of the two smoothed images; and (4) additional horizontal smoothing with another threshold C_a. Although it is conceptually simple, this algorithm requires scanning the image four times. WISDOM++ implements a variant that scans the image only twice, with no additional cost (Shih and Chen 1996). Furthermore, the smoothing parameters C_v and C_a are adaptively defined depending on the spread factor, while C_h is set to one-tenth of the number of columns in the reduced bitmap (that is, 62) (see Fig. 2).

The segmentation algorithm returns blocks that may contain either textual or graphical information. In order to facilitate subsequent document processing steps it is important to classify these blocks according to content type. The classes used in WISDOM++ are: *text block*,

horizontal line, vertical line, picture (i.e., halftone images), and *graphics* (e.g., line drawings).

A method for the classification of blocks into text, graphic/halftone image, horizontal line or vertical line was proposed by Wong et al. (1982). It is based on four features[3]: (1) the height of a block; (2) the eccentricity of the rectangle surrounding the block; (3) the mean horizontal length of the black runs of the original data within each block; and (4) the ratio of the number of black pixels to the area of the surrounding rectangle. Three textural features proposed by Wang and Srihari (1989), namely short-run emphasis, long-run emphasis and extra-long-run emphasis, were proven to be useful to discriminate between text blocks with large, medium-sized and small letters. Both Wong et al. and Wang and Srihari use linear discriminant functions as block classifiers, while Fisher et al. (1990) apply a rule-based classification method using only geometric features. A common aspect of all these works is that they adopt a set of threshold values, which do not automatically adjust for different types of documents.

In WISDOM++, the classification of blocks is performed by means of a decision tree automatically built from a set of training examples (blocks) of the five classes. The choice of a "tree-based" method instead of the most common generalized linear models is due to its inherent flexibility, since decision trees can handle complicated interactions among features and give easily interpreted results. The numerical features used by the system to describe each block are the following:

1. *height*: height of the reduced image block;
2. *length*: length of the reduced image block;
3. *area*: area of the reduced image block *(height*length)*;
4. *eccen*: eccentricity of the reduced image block *(length/height)*;
5. *blackpix*: total number of black pixels in the reduced image block;
6. *bw_trans*: total number of black-white transitions in all rows of the reduced image block;
7. *pblack*: percentage of black pixels in the reduced image block *(blackpix/area)*;
8. *mean_tr*: average number of black pixels per black-white transition *(blackpix/bw_trans)*;
9. *F1*: short-run emphasis;
10. *F2*: long-run emphasis;
11. *F3*: extra-long-run emphasis[4].

A well-known decision tree learning system is C4.5 by Quinlan (1993). It is a *batch* learner, that is, it cannot change the decision tree when some blocks are misclassified unless a new tree is generated from scratch using an extended training set. However, to enable users to train the system on-line when they are dissatisfied with the classification made by a decision tree it is necessary to apply an *incremental* learning strategy, as done by ITI 2.0 (Utgoff 1994). This system, which has been embedded in WISDOM++, can operate in three different ways. In the *batch* mode it works like C4.5. In the *normal* operation mode it first updates the frequency counts associated to each node of the tree as soon as a new instance is received. Then it restructures the decision tree according to the updated frequency counts. In the *error-correction* mode, frequency counts are updated only in the case of misclassification of the new instance. The main difference between the two incremental modes is that the normal operation mode guarantees the building of the same decision tree independently of the order in which the examples are presented, while the error-correction mode does not.

[3] In Wong et al.'s work, only the first three features are actually used by the block classifiers.

[4] Computed using the following thresholds: T1=10 and T2=20 (Wang and Srihari 1989).

4 Knowledge-based detection of the layout structure

The result of the segmentation process is a list of classified blocks, corresponding to printed areas in the page image[5]. Each block is described by a pair of coordinates, namely top left-hand corner and bottom right-hand corner, and the class label. The number of blocks is generally less than a hundred; thus, a segmented page is certainly easier to manage than the original bitmap. However, this new representation is still too detailed for the subsequent processing steps, which can work more efficiently with representations at higher levels of abstraction.

In this case, abstraction is related to the perception of document images. The perceptual organization process that aims to detect structures among blocks is called the *layout analysis*. The result is a hierarchy of abstract representations of the document image and the *geometric* (or *layout*) *structure*. The leaves of the layout tree (lowest level of the abstraction hierarchy) are the blocks, while the root represents the whole document. In multi-page documents, the root represents a set of pages. A page may group together several layout components, called *frames*, which are rectangular areas of interest in the document page image. An ideal layout analysis should produce a set of frames, each of which can be associated with a distinct *logical* component, such as title and author of a scientific paper. In practice, however, a sub-optimal layout structure, in which it is still possible to distinguish the logical meaning of distinct frames, should be considered a good output of the layout analyzer.

The various approaches to the extraction of the layout structure can be classified in two distinct dimensions: (1) direction of construction of the layout tree (top-down or bottom-up); and (2) amount of explicit knowledge used during the layout analysis. As to the second dimension, Nagy, Kanai and Krishnamoorthy (1988) distinguish three levels of knowledge in the layout structure of a document:

– *Generic* knowledge (e.g., type base lines of a word are collinear).

[5] Note that the assumption that printed areas are rectangular encompasses the assumption that all text lines have no skew. This explains the need to evaluate the page skew and to rotate the image.

– *Class-specific* knowledge (e.g., no text line is lateral to a graphical object).
– *Publication-specific* knowledge (e.g., maximum type size is 22 points).

They observe that knowledge used in bottom-up layout analysis is necessarily different from that used for top-down processing: it is much less document specific. In addition, we note that knowledge used in top-down approaches is typically derived from the relations between the geometric and the logical structures of specific classes of documents. This is the case of page grammars (Nagy et al. 1992) and geometric trees (Dengel and Barth 1988), which are used to segment document images and simultaneously associate some layout components with the logical structure. In WISDOM++ this class-specific knowledge is solely required in the document classification and understanding steps and it is automatically learned from examples of documents, as explained in the next section.

LEX is an example of a bottom-up layout analysis system that exploits generic knowledge on west-style typesetting conventions to group basic blocks together into frames (Esposito et al. 1995). The layout analysis is done in two steps:

1. A *global* analysis of the document image in order to determine possible areas containing paragraphs, sections, columns, figures, and tables. This step is based on an iterative process, in which the vertical and horizontal histograms of text blocks are alternatively analyzed to detect columns and sections/paragraphs, respectively.
2. A *local* analysis of the document to group together blocks which possibly fall within the same area. Three perceptual criteria are considered in this step: *proximity*
(e.g., adjacent components belonging to the same column/area are equally spaced), *continuity* (e.g., overlapping components), and *similarity* (e.g., components of the same type, with an almost equal height).

Pairs of layout components that satisfy some of these criteria may be grouped together. Each layout component is associated with one of the following types: text, horizontal line, vertical line, picture, graphic, and mixed. When the constituent blocks of a logical component are homogeneous the same type is inherited by the logical component, otherwise the associated type is set to *mixed*.

The first Prolog version of LEX (Malerba et al. 1995) had the advantage of a straightforward representation and manipulation of declarative knowledge. In order to improve the run time, a more recent C++ version has been implemented and embedded in WISDOM++. In Fig. 3) the various steps of the layout analysis process are shown.

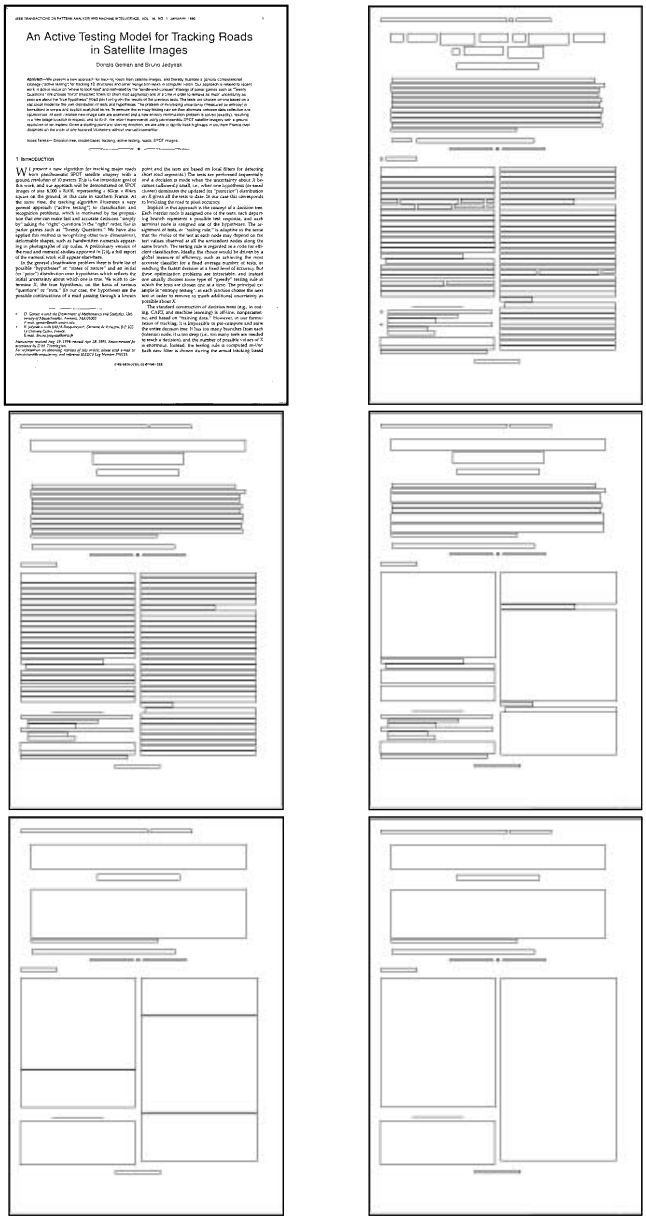


Fig. 3. A document image (*upper-left*) and the five levels of its layout structure: basic blocks, lines, set of lines, frame1, and frame2

5 Document classification and understanding

While the layout structure associates the content of a document with a hierarchy of layout objects, such as blocks, frames, and pages, the logical structure of the document associates the content with a hierarchy of logical objects, such as sender/receiver of a business letter, title/authors of a scientific article, and so on. The problem of finding the logical structure of a document can be reformulated as the problem of defining a mapping from the layout structure into the logical one. In WISDOM++ this mapping is currently limited to the association of a page with a document class (*document classification*) and the association of page layout components with basic logical components (*document understanding*). The mapping is built by matching the document description with both models of the classes of documents and models of the logical components of interest for each class.

Models are represented as rules. Typically such rules are handcoded for particular classes of documents (Nagy

et al. 1992), requiring fine-tuning and great human effort. In WISDOM++ rules are automatically generated by means of machine learning algorithms that induce them from a set of training examples, for which the final user has already defined the correct class and has specified the layout components with a logical meaning (logical components) (Esposito et al. 1999). In the current version of the system, the administrator has the responsibility of activating learning processes for different training sets.

In order to apply machine learning techniques to induce rules for layout-based document classification, it is important to define a suitable representation of the layout structure of training documents. In this work we confine ourselves to representing the most abstract level of the layout structure (*frame2*) and we deliberately ignore other levels, as well as their composition hierarchy. We prefer to describe both documents and rules in a first-order language. In this language, unary function symbols, called *attributes*, are used to describe properties of a single layout component (e.g., height and length), while binary predicate and function symbols, called *relations*, are used to express spatial relationships among layout components. An example of a page layout description, automatically generated by WISDOM++ for the layout shown in Fig. 3, is reported below:

part_of(X1,X2), part_of(X1,X3), ... , part_of(X1,X17),
width(X2)=290, width(X3)=94, ... , width(X17)=104,
height(X2)=6, height(X3)=7, ... , height(X17)=6,
type_of(X2)=hor_line, type_of(X3)=text,
type_of(X17)=text,
x_pos_center(X2)=168, x_pos_center(X3)=366, ... ,
x_pos_center(X17)=290,
y_pos_center(X2)=27, y_pos_center(X3)=27, ... ,
y_pos_center(X17)=783,
on_top(X2,X4), on_top(X3,X4), ... , on_top(X14,X15),
to_right(X15,X16), to_right(X2,X3), ... ,
to_right(X10,X11),
alignment(X2,X12)=only_left_col, ... ,
alignment(X10,X11)=only_middle_row

where the constant *X1* denotes the whole page, while the remaining constants *X2, X3, ... X17* denote distinct layout components at the frame2 level. It is noteworthy that both symbolic and numeric attributes are used to describe the page layout, thus requiring an extension of the machine learning algorithm originally designed for symbolic data alone. Indeed, some experimental results of the document image understanding problem (Malerba et al. 1997a) have led to the conclusion that mixed numeric/symbolic descriptions are essential for generating accurate models of logical structures. The following is an example of a rule, generated by the current version of the machine learning algorithm:

logic_type(X)=running_head ← y_pos_center(X)=18 ...
32, on_top(X,Y), on_top(Y,Z)

which means that an object X is a running head if its center is located in the upper part of the page (between rows 18 and 32) and is above another object Y, which is, in turn, above another object Z. A detailed description of the main issues and experimentally validated solutions for inductive learning of document classification and understanding models is reported in the paper by Esposito et al. (2000).

Once the logical structure of the document has been determined, WISDOM++ allows the user to set up the text extraction process, by selecting the logical components to which an OCR has to be applied. Finally, the system generates an HTML/XML version of the original document, as explained in the next section.

6 Transformation into HTML/XML formats

The simplest way to make processed documents accessible via the Web is to attach document images to HTML pages, after having converted bitmaps into a format supported by most browsers (e.g., GIF or JPEG). However, this approach presents at least three disadvantages. First, compressed raster images are still quite large and their transfer can be unacceptably slow. Second, the original document can only be viewed and not edited. Third, in the case of multipage documents, pages can be presented only in a sequential order, thus missing the advantages of a hypertext structure which supports document browsing. Therefore, it is important to transform document images into HTML/XML formats by aggregating all textual, graphical, layout and logical information extracted in the document analysis and understanding processes. Current commercial OCR technology is still far from performing this transformation effectively. The main reason is that OCR systems have insufficient information on the layout structure of documents and no information on the logical structure. Since knowledge of both layout and logical structures is actually available in WISDOM++ at the end of the document analysis and recognition process, it is possible to have qualitatively better results in terms of presentation and conveyed information (see Fig. 4). Henceforth, only the transformation into XML format will be presented, since it encompasses the generation of documents in HTML format.

6.1 XML document structures

An XML document has both a logical and a physical structure. The logical structure allows a document to be divided into named units and sub-units, called *elements*. The physical structure allows components of the document, called *entities*, to be named and stored separately, sometimes in other data files, so that information can be re-used, and non-XML data (e.g., images) can be included by reference. An XML processor is used to manage entities and combine them in a single data stream, both for validation by a *parser* and for accessing by the main application (see Fig. 5).

The most significant feature of XML is the concept of *D*ocument *T*ype *D*efinition (*DTD*), which *provides* a formal set of rules to define a logical document structure, *defines* the elements that may be used, and *dictates*

Fig. 4. Transformation of a document into HTML format as performed by a commercial OCR (*left*), and the transformation into XML format as performed by WISDOM++ (*right*). WISDOM++ renders a version of the document similar in appearance and content to the original document. Moreover, data on the affiliation of authors is not intermixed with the introduction text. WISDOM++ has associated the document with a style sheet specific for the class 'Transactions on Pattern Analysis and Recognition', which has been recognized during the document classification step. Although it is not visible, information on the logical structure of the document (running head, title, author(s), affiliation, abstract, index terms, and so on) is also reported in the XML file

where they may be applied in relation to each other. The declarations that comprise the DTD may be totally or partially stored at the top of each document that must conform to these rules (*internal* DTD), or may be alternatively stored in a separate data file (*external* DTD), which is referred to by a special instruction at the top of each document. WISDOM++ adopts the latter solution, which generates a distinct DTD for each document class. For instance, the DTD generated for any document of the class "Transactions on Pattern Analysis and Machine Intelligence" will be the following:

```
<!-- standard DTD file for tpami class -->
<!ELEMENT tpami
(abstract|affiliation|author|body|index-
term|page-number|running-head|title|undefined)*>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT index-term (#PCDATA)>
<!ELEMENT page-number (#PCDATA)>
<!ELEMENT running-head (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT undefined (#PCDATA)>
```

Each declaration conforms to the markup declaration format `<!.....>`. The keyword `ELEMENT` introduces an element declaration (in this case abstract, affiliation, undefined[6]) and specifies its allowed content. An element may have no content at all, may have a content of only text, of only child elements, or of a mixture of elements and text. In the example above, the content of the element `tpami` is a child element, which is structured, while all remaining elements can contain only text (*Parsable Character Data*, `PCDATA`). An attribute may be associated with a particular element to provide refined information on an element. Examples of attributes are the font size and the alignment. All the attributes are declared separately from the element, but are usually declared together in the *attribute list declaration*. In WISDOM++ the elements have no attribute, since information on font size, font weight, and so on is reported in a separate style sheet file (.css). It is also noteworthy that the DTD generated by WISDOM++ has no definition of elements, since our main goal is not to represent the layout structure explicitly, but to render the document similar in appearance to the original document, which can be achieved by means of XSL specifications, as explained later. In this way files to be transmitted through

[6] The element *undefined* refers to all those logical components of no specific interest for the application

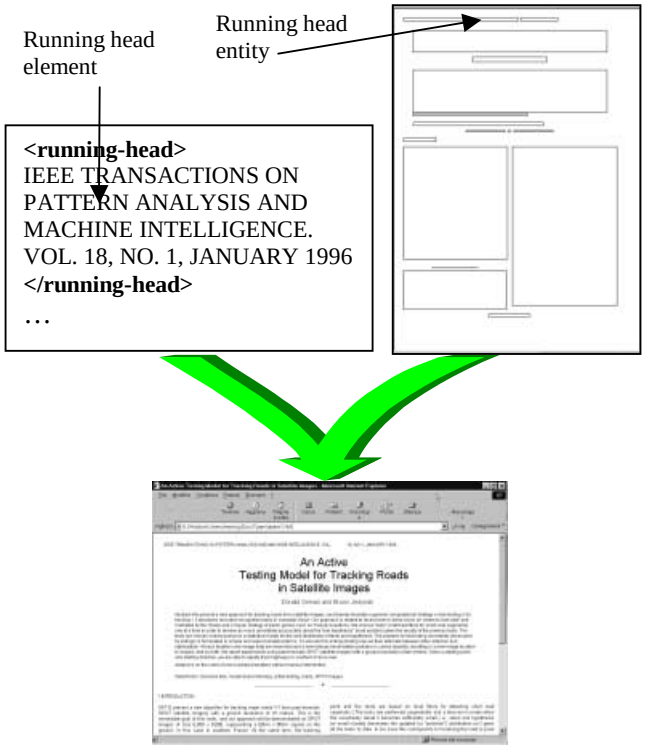


Fig. 5. Combination of the logical and physical structures in a single data stream rendered by an XML-enabled web browser

the Web are generally smaller, since they do not contain information on the hundreds of components in the document layout structure.

Once the DTD has been defined an instance of that document type can be generated and stored in an *.xml* file by respecting constraints defined by the set of rules in the DTD. An example of an *.xml* file, generated for the document in Fig. 3, is as follows:

```
<?xml-stylesheet href="tpami1.XSL"
type="text/xsl"?>
<!DOCTYPE tpami SYSTEM "tpami.DTD">
<tpami>
<running-head ID="id0"><paragraph>IEEE
TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE
INTELLIGENCE, VOL.
</paragraph></running-head>
<running-head ID="id1"><paragraph>18, NO.1,
JANUARY 1996
</paragraph></running-head>
<title><paragraph>An Active</paragraph>
<paragraph>
Testing Model for Tracking Roads</paragraph>
<paragraph>
in Satellite Images
</paragraph></title>
<author><paragraph>Donald Geman and Bruno
Jedynak
</paragraph></author>
<abstract ID="id4"><paragraph>Abstract-We
present a new
approach for tracking roads from
satellite images, and thereby illustrate a
general computational
strategy (∼ctive testing∼) for tracking 1 0
structures and otner
recognition tasks in computer vision. Our
approach is
 ...
</paragraph></body>
</tpami>
```

The first row specifies the name of the style sheet file (with extension *.xsl*), while the second row defines the DTD associated to the document class (file with extension .DTD). Text extracted with the OCR is intermixed with tags that define its logical structure (e.g., `<running-head>`, `<abstract>` and so on). The tag `<paragraph>` is used to mark lines returned by the OCR, information which is useful only for rendering purposes. This tag is not declared in the DTD since it is not considered a *logical* element from the user's viewpoint. The number of logical elements in the document instance corresponds to the number of *frame2* layout components to which the OCR has been applied. The element attribute ID is used to identify different instances of logical elements in the document. This is important for defining the proper rendering of each instance.

The XML specification includes a facility for physically isolating and separately storing any part of a document. Each unit of information is called an *entity* and each entity is assigned a name, so that it can be identified. The only entity to which an entity name is not assigned is the *document entity*. It is stored in a data file that is considered as representing the entire document. In simple cases the document entity may be the only entity (main program without sub-program), in more complex cases the document entity is used to position the call of other entities (main program with only subprograms). A declaration `<!ENTITY ... >` is required to announce the existence of an entity. No such declaration is reported in XML documents generated by WISDOM++, since they contain only one entity (document entity).

The content of an XML element, such as an abstract or a paragraph, has no explicit text *style* or *format*. Since XML language is not concerned with visualization aspects, it is necessary to specify the element rendering in a different language. *XSL* (*eXtensible Style Language*) is a language used for expressing style sheets. An XSL style sheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. The reference to an external style sheet is reported in the first row of the XML file produced by WISDOM++:

```
<?xml-stylesheet href="tpami1.XSL"
type="text/xsl"?>
```

An *XSL style sheet processor* accepts a document or data in XML and an XSL style sheet and produces the presentation of the XML source content that was intended by the designer of that style sheet. The presentation process involves two distinct steps: the transforma-

tion of the original XML source file (*tree construction*) and the interpretation of the transformed file to produce formatted results suitable for presentation (*formatting*). The process of formatting is performed by the formatter, which, in our case, is a rendering engine inside a browser. An example of a style sheet file generated for the document in Fig. 3, is as follows:

```
<?xml version='1.0'?>
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/TR/WD-xsl'>
<xsl:template match='/'>
<HTML>
<HEAD>
<TITLE>An Active Testing Model for Tracking Roads
in Satellite Images</TITLE>
<LINK rel="stylesheet" href="tpami.css"></LINK>
</HEAD>
<BODY TEXT="BLACK" BGCOLOR="WHITE">
<TABLE WIDTH='100%' BORDER='0'>
<TR>
<TD WIDTH='54%' VALIGN='TOP'
CLASS="running-head"><BR/>
<xsl:for-each
select="tpami/running-head[@ID='id0']/paragraph">
<xsl:if test='TAB'>     </xsl:if>
</xsl:for-each>
</TD>
 ...
</HTML>
</xsl:template>
</xsl:stylesheet>
```

The transformation is based on template rules (tag `<xsl:template>`), each rule specifying how an XML element should be transformed. In the above example, only one rule is defined to transform the whole XML file into an HTML document, by adding graphical components and by arranging elements into tables as well as lines into table entries. Following W3C recommendations, the selection of font, font-size, and alignment is not specified in the XSL file, but in the *Cascading Style Sheet* file (*.css*), which is unique for each class of documents. It is noteworthy that WISDOM++ does not rely on style information extracted by the OCR, but exploits the result of the classification process to associate a *.css* file to an HTML/XML document. For instance, the CSS file associated to the document in Fig. 3 will be:

```
TD {font: 7pt Arial; text-align: justify;}
TD.title {font-size: 19pt; text-align:
center;}
TD.author {font-size: 10pt; text-align:
center;}
TD.running-head {font-size: 7pt; text-align:
center;}
TD.abstract {font-size: 7pt; text-align:
left;}
TD.affiliation {font: 7pt Times New Roman;
font-posture: italic; }
TD.index-term {font-size: 7pt;}
TD.body {font: 8pt Times New Roman;}
TD.page-number {font-size: 7pt;}
BR {font-size: 3pt;}
```

WISDOM++ helps users to define a CSS file for each class of documents.

6.2 Generation of the HTML/XML files

The previous section illustrates the distribution into different files of several pieces of information extracted by WISDOM++, namely the layout and logical structures, the pictorial content of some graphical layout components, the textual content as well as the text format of some logical components. However, the document conversion into structured formats (HTML/XML) is not straightforward, since a number of factors should be considered in order to render the converted document as similar as possible to the original document image.

The system makes use of document layout information for conversion into structured formats. In particular, *frame2* and *lines* are the only two levels of the layout structure that are actually useful. The former defines the layout to be reproduced in a Web document, while the latter helps to format the text associated to some *frame2* components. Layout-based conversion into HTML/XML format is a two-phase process:

1. Creation of a hierarchy of HTML tables that accurately represent the highest level of the layout structure.
2. Formatting the text associated to some *frame2* components and linking the bitmaps extracted for the others.

The first step is indispensable, since neither HTML not XML allow coordinate-based positioning of text and images on the screen. The page layout is rendered by means of nested tables, which are generated according to the following procedure:

```
convert_column(column)
Determine rows in the column and generate an HTML table with as many rows
For each row with multiple frame2 components:
    Determine columns in the row and generate an HTML table with as many columns
    For each column c
        convert_column(c)
```

This procedure performs recursive cuts of the horizontal and vertical histograms, computed on the basis of *frame2* components. At the first call the layout of the whole document is passed. For the document in Fig. 3 this procedure determines a first row with two cells, one for the running head and one for the page number, which will be described by the following fragment of HTML code:

```
<TABLE WIDTH='100%' BORDER='0'>
<TR>
<TD WIDTH='54%' VALIGN='TOP'
CLASS="running-head"><BR>
<CENTER>IEEE TRANSACTIONS ON PATTERN ANALYSIS
AND MACHINE INTELLIGENCE, VOL.</CENTER>
</TD>
<TD WIDTH='1%'></TD>
```

```
<TD WIDTH='17%' VALIGN='TOP'
CLASS="running-head"><BR>
<CENTER>18, NO.1, JANUARY 1996</CENTER>
</TD>
```

The attribute WIDTH denotes the relative width of a table (row/column) and it is computed to respect the original size of *frame2* components. The content of table entries (tag <TD>) can be a text, a figure or a horizontal/vertical line. For instance, the seventh row of the table generated for the document in Fig. 3 contains the horizontal line that separates the index term from the introduction:

```
<TABLE WIDTH='100%' BORDER='0'>
<TR>
<TD WIDTH='29%'></TD>
<TD WIDTH='18%' VALIGN='TOP'><BR><HR></TD> left
horizontal line
<TD WIDTH='2%'></TD>
<TD WIDTH='1%' VALIGN='TOP'><BR>
<IMG SRC="tpami1j8.jpg"></TD>  diamond image
<TD WIDTH='1%'></TD>
<TD WIDTH='18%' VALIGN='TOP'><BR><HR></TD>
right horizontal line
<TD WIDTH='31%'></TD>
</TR>
</TABLE>
```

Once the hierarchy of HTML tables has been built it is necessary to format each table entry with textual content. This task is performed by the OCR when the output is already saved in HTML format, while it is done by WISDOM++ when the output is plain ASCII text. In the latter case the straightforward copy of the ASCII text would result in a loss of the original formatting of the document (e.g., arrangement of text in paragraphs, indentation, and centering). In order to recover the original formatting it is necessary to match text lines read by the OCR with *lines* in the layout structure (see Fig. 6). Generally, only a partial matching can be found, due to possible errors in the segmentation and layout analysis made by either WISDOM++ or the OCR. For instance, there is only one line for the section title in Fig. 6, while the OCR returns two text lines (each line ending with ¶). On the contrary, the first two lines of the initial paragraph are merged during the segmentation process, while the OCR returns two text lines. In WISDOM++ the optimal matching is found by means of a heuristic search. Additional heuristics are also used to decide when a text line should be indented or centered, as well as when a carriage return (¶) detected by the OCR should be transformed into a break tag (
) in HTML.

7 Benchmarking of WISDOM++

In order to assess the absolute performance of the document analysis system, we will follow the evaluation method proposed by Märgner et al. (1997). Attention is focused on the document analysis functions alone, since an assessment of the system performance for the document classification and understanding steps is out of the scope of this paper (see Esposito et al. (2000) for more details).

By considering WISDOM++ as a chain of modules, we intend to evaluate the performance of some of them, namely the skew evaluation module and the block classification module. A set of 112 real, single-page documents have been considered as input data. The set is the same as that used in a previous study of the document classification and understanding problems (Esposito et al. 1999). The documents are distributed as follows: thirty are the first pages of articles appearing in the proceedings of the International Symposium on Methodologies for Intelligent Systems (*ISMIS94*)[7], twenty-eight are the first pages of articles published in the proceedings of the 12th International Conference on Machine Learning (*ICML95*), thirty-four are the first page of articles published in the *IEEE Transactions on Pattern Analysis and Machine Intelligence* (*TPAMI*)[8], and twenty are documents of different types or published in other proceedings, transactions, and journals (*Reject*). The text is organized into one column for the first class of documents and into two columns for the second and third classes. No uniform layout can be detected for the documents in the fourth class. The documents may contain formulae, drawings, halftone images and other graphical elements (e.g., vertical/horizontal lines).

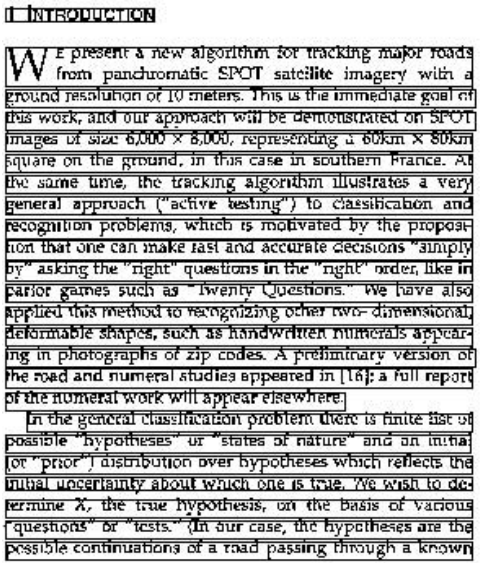
Benchmarking of the deskew algorithm is performed in two different ways. First, the optimal output (*ground truth*) is determined by a human expert, who decides which is the right skew angle of the scanned document image. This *real* assessment may contain some imprecision, that we tried to reduce by providing the expert with two effective tools in the interface: zooming functions and a straight-edge. The latter aims at an error-free *ideal* assessment through artificial rotations of the document image for some known angles. In both cases, the *evaluation function* computes the error made with respect to the real/ideal assessment value.

The results on the real assessment are reported in Table 1, where all measurements are absolute values of angles expressed in degrees. The following observations should be made. Documents have been scanned by a careful user, who placed documents on the flat surface of a scanner with a skew of less than three degrees. The smallest non-null skew angle that can be detected for A4 documents is 0.023°, which corresponds to a tilt of one pixel in a bitmap with 2,496 columns. For single-column documents (ISMIS94) the error made by WISDOM++ is two pixels on average, while for documents organized in two columns (ICML95 and TPAMI) it is about nine pixels on average.

Document images correctly rotated by the expert are used in the second benchmarking of the deskew algorithm. Each of them is rotated at various degrees (i.e., ±0.5°, ±1.0°, ±1.5°, ±2.0°, ±3.0°, ±4.0°, ±5.0°, ±6.0°, ±7.0°), using the rotation algorithm implemented in

[7] Published by Springer-Verlag in the series *Lecture Notes in Artificial Intelligence*, Vol. 869

[8] Period January-June 1996



1 INTRODuCTION¶
¶
XAT E present a new algorithm for tracking major roads¶
VY from panchromatic SPOT satellite imagery with a¶
ground resolution of 10 meters. This is the immediate goal of¶
this work, and our approach will be demonstrated on SPOT¶
images of size 6,000 x 8,000, representing a 60km x 80km¶
square on the ground, in this case in southern France. At¶
the same time, the tracking algorithm illustrates a very¶
general approach ("active testing") to classification and¶
recognition problems, which is motivated by the proposi-¶
tion that one can make fast and accurate decisions "simply¶
by" asking the "right" questions in the "right" order, like in¶
parlor games such as "Twenty Questions." We have also¶
applied this method to recognizing other tw~ dimensional, ¶
deformable shapes, such as handwritten numerals appear-¶
ing in photographs of zip codes. A preliminary version of¶
the road and numeral studies appeared in [16]; a full report¶
of the numeral work will appear elsewhere. ¶
In the general classification problem there is finite list of¶
possible "hypotheses" or "states of nature" and an initial¶
(or "prior") distribution over hypotheses which reflects the¶
initial uncertainty about which one is true. We wish to d~¶
termine X, the true hypothesis, on the basis of various¶
"questions" or "tests." (In our case, the hypotheses are the¶

Fig. 6. Lines detected in a layout structure (*left*) and text lines extracted by the OCR (*right*)

Table 1. Errors made with respect to the real assessment defined by an expert

	Ismis	*Tpami*	*Icml*	*Reject*
Average error	0,0470	0,2166	0,2045	0,1120
Average Real Skew	0,3115	0,4530	0,5130	0,3659
Standard Deviation	0,2827	0,3489	0,5121	0,3896

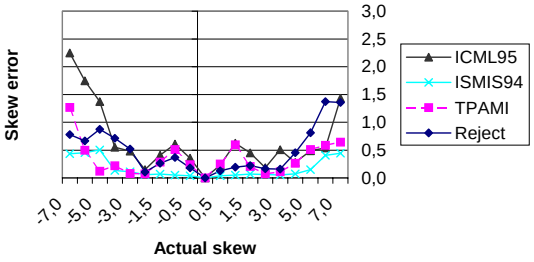


Fig. 7. Average absolute error of the skew angle for the four classes of documents

WISDOM++. The absolute errors averaged in all documents of a class are reported in Fig. 7.

A comparison with other published results is not easy. A similar benchmarking was performed by Smith (1995), whose best results seem to be similar to ours. Our experimental design allows us to observe that the skew estimation procedure gives a good performance for single-column documents, but is not always reliable for documents organized in two or more columns. This limit is more evident with clockwise rotations, and is generally due to the difficulty in selecting a good sample region. Moreover, the error generally increases with the size of the skew angle, so that for a relatively large tilt it would be necessary to repeat the deskew process more than once. As to the time performance, which is a critical factor for a real-time system, it is always lower than 0.41 s for a Pentium PC 200MMX.

In order to test the performance of the block classifier, the set of documents has been split into a training

Table 2. Characteristics of the learned decision trees

	Size Kb	*No. Nodes*	*No. leaves*	*No. incorporated examples*
Batch	24,320	229	114	9,429
Pure Error-correction	982	159	79	277
Mixed	13,150	235	117	4,545+125

set (70%) and a test set (30%), according to a stratified random sampling. The number of training blocks is 9,429, while the number of test blocks is 3,176. Three experiments have been organized. ITI 2.0 has been trained in the batch mode in the first experiment, pure error-correction mode in the second, and mixed incremental-/error-correction mode in the third (incremental mode for the first 4,545 examples and error-correction mode for the remaining 4,884).

The main characteristics of the learned trees are reported in Table 2, where the last column refers to the number of examples stored in the nodes of the induced trees.

The decision tree built in the batch mode takes about 24 Mb, since all instances have to be stored, while the decision tree obtained in the error-correction mode requires only 982 kb. The total number of instances incorporated in the third experiment is 4,670, where 4,545 are the training examples used in the first incremental step, while the remaining 125 have been incorporated in the subsequent error-correction step. Nevertheless, this difference in tree size corresponds to a very small difference in predictive accuracy estimated on the independent test set (see Table 3). This justifies the use of the decision tree, developed according to the error-correction operation mode, in many practical situations.

The learned trees are not shown in this paper because they are too large. We limit ourselves to reporting that the set of features actually used in the decision tree built by the pure error-correction procedure does not contain

Table 3. Predictive accuracy of the learned decision trees

	ISMIS 94	*ICML 95*	*TPAMI*	*Reject*	*Total*
Batch	95.78	97.74	98.26	97.00	97.48
Pure Error-correction	97.05	98.39	98.01	95.06	97.45
Mixed	95.99	97.63	98.18	97.35	97.51

two features, namely the width of the block and F3. This omission is probably due to the documents considered in this benchmarking, which have few occurrences of large text blocks better identified by the feature emphasizing long runs.

The benchmarking of the layout analysis process is more elaborate, since the extracted layout depends on a number of subsequent processing steps, namely adaptive segmentation, global layout analysis, and detection of lines, set of lines, *frame1* components and *frame2* components. In order to evaluate the result of one of these processing steps, it is necessary to provide the system with the ideal (or at least, the best) input data and to define the ground truth, that is, the ideal output of that step.

In the evaluation of the adaptive segmentation algorithm the ideal input is a deskewed document, while we defined as ground truth a set of blocks such that: (1) each block encloses homogeneous data (single characters, words or text lines of a column, horizontal/vertical lines, images, and graphics); (2) no block encloses pure noise, such as inkspots. Experimental results on the set of 112 documents are reported in Table 4.

Most of the errors are due to "noisy" blocks. A small percentage of the error rate is due to large blocks that group several text lines (i.e., the threshold C_v is high) or text lines of adjacent columns (i.e., the threshold C_a is high). It is noteworthy that changes made to both C_v and C_a do not entail a total recovery of all errors.

Experimental results for the global layout analysis are reported in Tables 5 and 6. They refer to errors observed in the detection of columns and sections, respectively. In this case the ideal input is the corrected result of the segmentation algorithm.

The ground truth defined for the line detection step is based on the following criteria: (1) each line in the layout structure should include a text line in the document; (2) contiguous horizontal and vertical lines extracted by the segmentation algorithm should be merged together. Experimental results are given in Table 7. Most of the errors are due to both block misclassification and mistakes in the global layout analysis. They have been corrected by changing the block classification or the segmentation thresholds C_v and C_a.

The ground truth defined for the detection step of the set-of-lines is based on the following criterion: each set-of-lines should include justified text lines of the same paragraph. Experimental results are reported in Table 8. In addition, in this case, errors are caused by block misclassification and erroneous global layout analysis.

The ground truth defined for the *frame1* detection step is based on the following criteria: (1) a block includes a set of centered text lines concerning the same logical component (title, authors, etc.); (2) a block includes a set of text lines aligned by the left (right) column, which concern the same logical component (abstract, affiliation, footnotes, etc.); (3) a block includes a set of text lines in a paragraph; and (4) a block includes a single text line regarding a page number or a section title. Experimental results are reported in Table 9.

Finally, the ground truth at the *frame2* level is the ideal decomposition of a page into layout components, such that each of them can be associated to at most one logical label and the conditions reported for frame1 are satisfied. Experimental results are reported in Table 10.

These experimental results provide a quantitative evaluation of the correctness of individual steps in the document image analysis. However, errors may accumulate at each processing step, as shown in Fig. 8. In this example, an error in the detection of sections generates an erroneous grouping at the level of lines, that in turn affects the generation of *frame2* components. The final effect is that both the abstract and the body of the paper are grouped together and the rules for document image understanding cannot correctly map the layout structure into the logical structure. Figure 8 also illustrates some examples of noisy blocks erroneously detected by the segmentation algorithm and carried forward up to the *frame2* level. In both cases, user intervention on the extracted layout is required in order to recover the correct layout structure.

8 Conclusions

In this paper a novel document processing system has been presented. The system is complete and effectively supports the transformation of printed documents into HTML/XML format. It exploits the layout structure extracted in the layout analysis step, as well as the result of the various classifiers for blocks, documents, and logical components automatically generated by symbolic machine learning tools. A benchmarking of some processing steps has also been reported. In particular, the deskew algorithm, which operates in two steps, gives a good performance for single column documents, while there is still a margin for improvement in documents with a more complex layout structure. Deskewed documents are segmented by means of a fast RLSA with an adaptive parameter definition. The average percentage of errors observed in the benchmarking of the adaptive segmentation algorithm is relatively small and without serious consequences for the subsequent layout analysis steps. A high predictive accuracy has been observed for the block classifier, which is a decision tree automatically induced from a set of training examples. More abstract layout components are found by means of a bottom-up, knowledge-based layout analysis. The promising results reported in the benchmarking for the highest level in the layout structure empirically support the idea of using it as the basis for document classification, document under-

Table 4. Benchmarking of the adaptive segmentation algorithm

Class	*No. blocks*	*No. errors*	*Perc. errors*	*No. blocks after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	3931	135	3.4	3691	14	0.3
Tpami	4477	198	4.4	4349	48	1.1
Ismis	1677	171	10.2	1485	4	0.3
Reject	2520	159	6.3	1654	15	0.9
Total	12605	663	5.3	11179	81	0.7

Table 5. Benchmarking of the global layout analysis process: column detection

Class	*No. columns*	*No. errors*	*Perc. errors*	*No. columns after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	64	7	10.9	61	4	6.5
Tpami	77	8	10.4	72	5	6.9
Ismis	31	2	6.4	30	1	3.3
Reject	35	7	20	32	5	15.6
Total	207	24	11.6	195	15	7.7

Table 6. Benchmarking of the global layout analysis process: section detection

Class	*No. sections*	*No. errors*	*Perc. errors*	*No. sections after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	398	8	2	396	6	1.5
Tpami	544	26	4.8	538	20	3.7
Ismis	244	4	1.6	240	3	1.2
Reject	227	20	8.8	223	18	8.1
Total	1413	58	4.1	1397	47	3.4

Table 7. Benchmarking of the line detection step

Class	*No. lines*	*No. errors*	*Perc. errors*	*No. lines after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	2421	26	1.1	2409	14	0.6
Tpami	2914	66	2.3	2873	42	1.4
Ismis	1195	8	0.7	1191	1	0.1
Reject	1214	23	1.9	1214	20	1.6
Total	7744	123	1.6	7687	77	1.0

Table 8. Benchmarking of the set-of-line detection step

Class	*No. set-of-lines*	*No. errors*	*Perc. errors*	*No. set-of-lines after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	708	23	3.2	693	11	1.6
Tpami	1527	30	2.0	1494	18	1.2
Ismis	507	8	1.6	500	3	0.6
Reject	728	23	3.2	702	8	1.1
Total	3470	84	2.4	3389	40	1.2

Table 9. Benchmarking of the frame1 detection step

Class	*No. frames*	*No. errors*	*Perc. errors*	*No. frames after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	408	12	2.9	402	6	1.5
Tpami	728	25	3.4	695	12	1.7
Ismis	265	12	4.5	258	1	0.3
Reject	351	20	5.7	339	9	2.7
Total	1752	69	3.9	1694	28	1.7

Table 10. Benchmarking of the frame2 detection step

Class	*No. frames*	*No. errors*	*Perc. errors*	*No. frames after correction*	*No. errors after correction*	*Perc. errors after correction*
Icml	316	27	8.5	310	17	5.5
Tpami	556	24	4.3	531	6	1.1
Ismis	181	10	5.5	184	2	1.1
Reject	236	24	10.2	229	10	4.4
Total	1289	85	6.6	1254	35	2.8

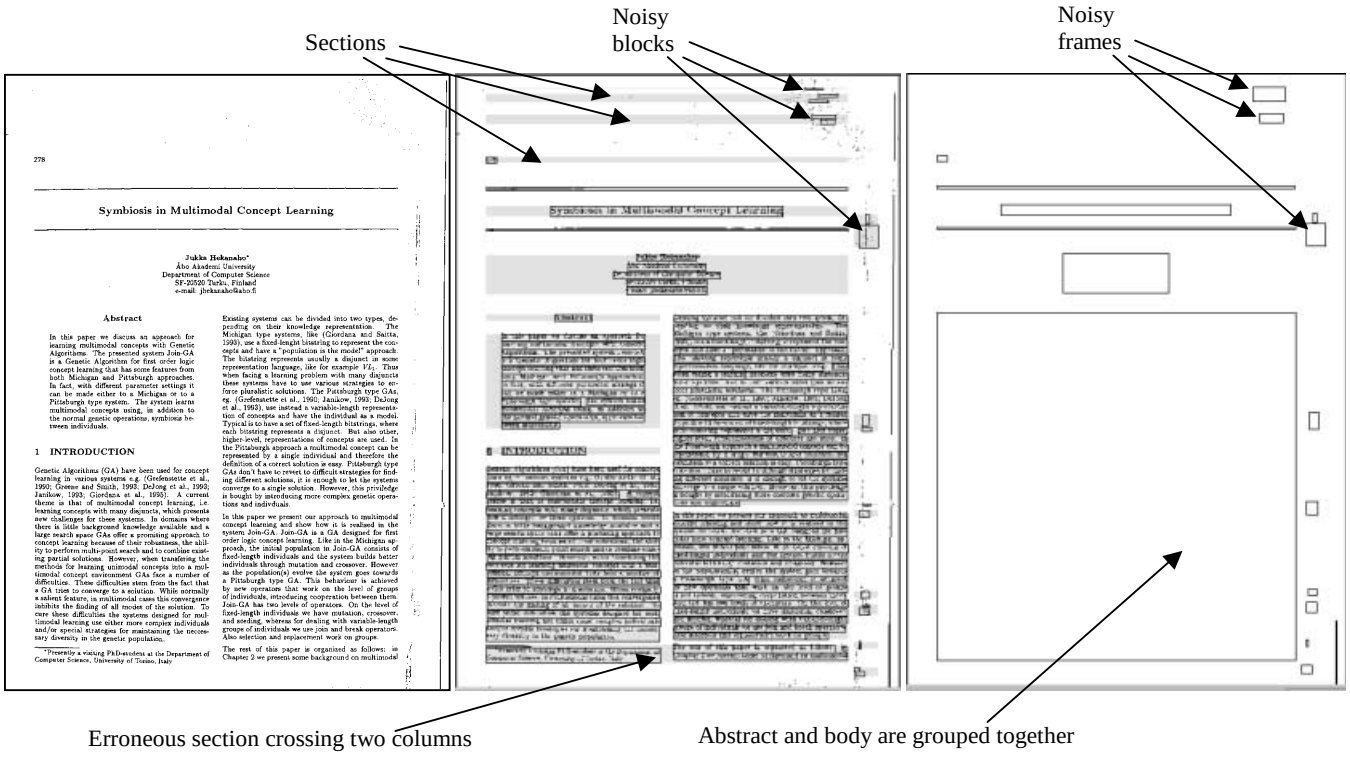


Fig. 8. Original document of the class ICML95 (*left*); basic blocks extracted by the segmentation algorithm and sections detected by the global layout analysis (*center*); final frame2 level (*right*)

standing, and transformation into HTML/XML format. Currently, we are investigating the possibility of inducing class-specific knowledge to be used in the global analysis process. By correcting the results of the global analysis with either splitting or grouping operations it is possible to generate a set of training examples for a learning system that can automatically induce class-specific global analysis rules. Moreover, we plan to extend the hypertext structure of the generated HTML/XML documents by creating indexes and adding links to bibliographic references.

Acknowledgements. The authors would like to thank Francesco De Tommaso, Dario Gerbino, Ignazio Sardella, Giacomo Sidella, Rosa Maria Spadavecchia, and Silvana Spagnoletta for their contribution to the development of WISDOM++. We are also grateful to Lynn Rudd for her help in re-reading the paper.

References

O. Altamura, F. Esposito, D. Malerba: WISDOM++: an interactive and adaptive document analysis system. In: Proc. 5th Int. Conf. on Document Analysis and Recognition, pp. 366–369, IEEE Computer Society, Los Alamitos, Calif., USA, 1999

A. Antonacopoulos: Local skew angle estimation from background space in text regions. In: Proc. 4th Int. Conf. on Document Analysis and Recognition, pp. 684–688, IEEE Computer Society, Los Alamitos, Calif., USA, 1997

H.S. Baird: The skew angle of printed documents. In: Proc. Conference of the Society of Photographic Scientists and Engineers, pp. 14–21, 1987

A. Dengel, G. Barth: High-level document analysis guided by geometric aspects. Int. J. Pattern Recognition. Artif. Intell., 2: 641–655, 1988

A. Deutsch, M.F. Fernandez, D. Florescu, A. Levy, D. Suciu: A query language for XML. In: Proc. 8th International World Wide Web Conference, pp. 77–91, 1999

F. Esposito, D. Malerba, G. Semeraro: Multistrategy learning for document recognition. Appl. Artif. Intell. 8(1): 33–84, 1994

F. Esposito, D. Malerba, G. Semeraro: A knowledge-based approach to the layout analysis. In: Proc. 3rd Int. Conf. on Document Analysis and Recognition, pp. 466–471, IEEE Computer Society, Los Alamitos, Calif., USA, 1995

F. Esposito, D. Malerba, F.A. Lisi: Machine learning for intelligent document processing: the WISDOM system. In: Z.W. Ras, A. Skowron (eds.) Foundations of intelligent systems, Lecture Notes in Artificial Intelligence, vol. 1609. Springer, Berlin Heidelberg New York, pp. 103–113, 1999

F. Esposito, D. Malerba, F.A. Lisi: Machine learning for intelligent processing of printed documents. J. Intell. Inf. Syst. 14(2/3): 175–198, 2000

J.L. Fisher, S.C. Hinds, D.P. D'Amato: A rule-based system for document image segmentation. In: Proc. 10th Int. Conf. on Pattern Recognition, pp. 567–572, IEEE Computer Society, Los Alamitos, Calif., 1990

O. Hitz, L. Robadey, R. Ingold: Using XML in Document Recognition. In: Proc. International Workshop on Document Layout Interpretation and its Applications, Bangalore, India, 1999

D. Hix, H.R. Hartson: Developing user interfaces: ensuring usability through product and process. Wiley, London, 1993

D. Malerba, G. Semeraro, E. Bellisari: LEX: a knowledge-based system for the layout analysis. In: Proc. 3rd Int. Conf. on the Practical Application of Prolog, pp. 429–443, 1995

D. Malerba, F. Esposito, G. Semeraro, S. Caggese: Handling continuous data in top-down induction of first-order rules. In: M. Lenzerini (ed.), AI*IA 97: Advances in artificial intelligence, Lecture Notes in Artificial Intelligence, vol. 1321. Springer, Berlin Heidelberg New York, pp. 24–35, 1997a

D. Malerba, F. Esposito, G. Semeraro, L. De Filippis: Processing paper documents with WISDOM. In: M. Lenzerini (ed.), AI*IA 97: advances in artificial intelligence, Lecture Notes in Artificial intelligence, vol. 1321. Springer, Berlin Heidelberg New York, pp. 439–442, 1997b

V.F. Märgner, P. Karcher, A.-K. Pawlowski: On benchmarking of document analysis systems. In: Proc. 4th Int. Conf. on Document Analysis and Recognition, pp. 331–336, IEEE Computer Society, Los Alamitos, Calif., USA, 1997

G. Nagy, J. Kanai, M. Krishnamoorthy: Two complementary techniques for digitized document analysis. In: Proc. ACM Conference on Document Processing Systems, Santa Fe, New Mexico, pp. 169–176, 1988

G. Nagy, S. Seth, M. Viswanathan: A prototype document image analysis system for technical journal. IEEE Comput. 25(7): 10–22, 1992

J.R. Quinlan: C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, Calif., 1993

S. J. Simske: The use of XML and XML-data to provide document understanding at the physical, logical and presentational Levels. In: Proc. International Workshop on Document Layout Interpretation and its Applications, Bangalore, India, 1999

R. Smith: A simple and efficient skew detection algorithm via text row accumulation. In: Proc. Third Int. Conf. on Document Analysis and Recognition, pp. 1145–1148, IEEE Computer Society, Los Alamitos, Calif., USA, 1995

F.Y. Shih, S.-S. Chen: Adaptive document block segmentation and classification. IEEE Trans. Syst. Man Cybern. – Part B 26(5): 797–802, 1996

P.E. Utgoff: An improved algorithm for incremental induction of decision trees. In: Proc. 11th Int. Conf. on Machine Learning, Morgan Kaufmann, San Francisco, Calif., USA, 1994

C.L. Yu, Y.Y. Tang, C.Y. Suen: Document skew detection based on the fractal and least squares method. In: Proc. 3rd Int. Conf. on Document Analysis and Recognition, pp. 1149–1152, IEEE Computer Society, Los Alamitos, Calif., USA, 1995

D. Wang, R.N. Srihari: Classification of newspaper image blocks using texture analysis. Comput. Vision Graphics Image Proc. 47: 327–352, 1989

Y. Wang, T. Phillips, R. Haralick: From image to SGML/XML representation: one method. Proc. International Workshop on Document Layout Interpretation and its Applications, Bangalore, India, 1999

E. Wenger: Glass-box technology: merging learning and doing. IRL Research Abstract 1, Institute for Research on Learning, Palo Alto, Calif., USA, 1988

K.Y. Wong, R.G. Casey, F.M. Wahl: Document analysis system. IBM J. Res. Dev. 26(6): 647–656, 1982

M. Worring, A.W.M. Smeulders: Content-based Internet access to scanned documents. Int. J. Doc. Anal. Recognition 1(4), 1999

World Wide Web Consortium *W3C*:
Extensible Markup Language ver1.0
http://www.w3.org/TR/1998/REC-xml-19980210
Extensible Stylesheet Language ver.1.0
http://www.w3.org/TR/1998/WD-xsl-19980818
XSL Transformation ver. 1.0
http://www.w3.org/TR/1999/07/WD-xslt-19990709

Oronzo Altamura is an Assistant Professor of Computer Science at the University of Bari in the Department of Informatics. He received a "laurea" degree in Physics from the University of Bari in June 1973. For many years he collaborated with the Institute of Physics of the University of Bari. Initially his research activity concerned signal processing, digital system design, fault tolerant systems. Then his research interests moved to computer assisted instruction (CAI), intelligent CAI, intelligent tutoring systems (ITS). He published same papers an these topics in international journals and proceedings. Currently he collaborates with the Machine Learning group of the University of Bari an document image segmentation, skew detection, and intelligent document processing.

Floriana Esposito is a Full Professor of Computer Science and responsible for LACAM (Laboratory for owledge Acquisition and Machine Learning). Since 1997 she has been the director of the Interdepartmental Center for Logic and Applications of the University of Bari, and Dean of the Faculty of Computer Science of the University of Bari. Currently. Her main research interests are in similarity based learning, multistrategy learning and incremental learning. The major field of application concerns document analysis and understanding: with her research group she has been involved in the ESPRIT project 5203 INTREPID (INnovative Techniques for REcognition and Processing of Documents). Currently the group is involved in the ESPRIT Project 29159 CONCERTO (CONCEptual indexing, querying and ReTrieval Of digital documents) and the IST project 20882 COLLATE (COLLaboratory for Annotation, Indexing and Retrieval of Digitized Historical Archive Material). She is author of more than one hundred papers which have been published in international journals and proceedings; she is an the directorial board of the Italian Association for Artificial Intelligence and is currently responsible for the national Machine Learning Group. She has been in the scientific committees of many international Conferences (ECML'94-98-2000, AI*IA'93-95-97-99, ECAI'96, ICDAR'97-99, ISMIS 2000, MSL'98-99-2000, ICML'99); she organized and chaired different conferences and workshops in the field of Machine Learning.

Donato Malerba is an Associate Professor of Computer Science at the University of Bari, Italy. In 1987 fie received a "laurea" degree in Informatics and in 1992 he was assistant specialist at the Institute of Computer Science, University of California, Irvine. For the past decade he has been active in machine learning and its applications to intelligent document processing, knowledge discovery in databases, digital map interpretation, and intelligent interfaces. He has published more than sixty papers in refereed journals and conferences. He is involved in many national and European projects an data mining, machine learning and document processing. He has served as co chair of several workshops, as a lecturer in the ICDAR'99 tutorial an "Learning in Document Analysis and Understanding", and as a program committee member of the International Conference an Machine Learning (ICML'96, ICML'99), of the 12th International Symposium an Methodologies for Intelligent Systems (ISMIS 2000), and of the International Workshop an Machine Learning and Data Mining in Pattern Recognition (MLDM 2001). He is a member of the International Association of Pattern Recognition (IAPR).