

Recognition of Common Areas in a Web Page Using Visual Information: a possible application in a page classification

Miloš Kovačević¹, Michelangelo Diligenti², Marco Gori², Marco Maggini², Veljko Milutinović³

¹School of Civil Engineering, University of Belgrade, Serbia
milos@grf.bg.ac.yu

²Dipartimento di Ingegneria dell'Informazione, University of Siena, Italy
{diligmic, maggini, marco}@dii.unisi.it

³School of Electrical Engineering, University of Belgrade, Serbia
vm@etf.bg.ac.yu

Abstract

Extracting and processing information from web pages is an important task in many areas like constructing search engines, information retrieval, and data mining from the Web. Common approach in the extraction process is to represent a page as a “bag of words” and then to perform additional processing on such a flat representation. In this paper we propose a new, hierarchical representation that includes browser screen coordinates for every HTML object in a page. Using visual information one is able to define heuristics for recognition of common page areas such as header, left and right menu, footer and center of a page. We show in initial experiments that using our heuristics defined objects are recognized properly in 73% of cases. Finally, we show that a Naive Bayes classifier, taking into account the proposed representation, clearly outperforms the same classifier using only information about the content of documents.

1. Introduction

Web pages are designed for humans! While previous sentence is more than obvious, still many machine learning and information retrieval techniques for processing web pages do not utilize implicit visual information contained in the HTML source. By visual information we assume positions of HTML objects in the browser window. For example, one can say that certain image is on the top left corner of the screen or that the most informative paragraph is in the center of the page and it occupies the area of 100x200 pixels.

Where this kind of information can be useful? Consider the problem of feature selection in document (web page) classification. There are several methods to perform feature selection process like Information Gain [1] or TF-IDF (term frequency – inverse document frequency) [2]. In both cases we try to estimate what are the most relevant words that describe document D i.e. the best vector representation of D that will be used in

classification process. Assuming that web pages are designed for visual sense, we can argue that some words represent noise with respect to page topic if they belong to menu, banner link or perhaps page footer. That noise can be misleading for classifiers. Also, we can suppose words that belong to the central part of the page (screen) carry more information than words from the down right corner. Hence there should be a way to weight differently words from different layout contexts. At present moment in classic algorithms, positions of words and their spanning areas are not considered at all!

Let us mention another problem – designing efficient crawling strategy for focused search engines. Given a specific topic T and starting set of pages S , it is necessary to find as more T on-topic pages as possible in a predefined number of steps. By step is meant visiting (and downloading and indexing) a page reachable from S following hyperlinks from pages in S . In other words it is important to estimate whether an outgoing link is promising or not. In [3][4] and [5] different techniques are described. In any case when crawler decides to take into account page for link expansion, all links from the page are inserted into the crawl frontier (links that are to be visited). But many of them are not interesting at all (i.e. “*this page is designed by XYZ*”). Sometimes links that belong to menus or footer are also misleading. Can we measure the importance of the link according to link position in the page (on the browser screen)? Links in the center of the page are probably more important than links in the down left corner. Also, we can calculate link density in some area of the page (screen) and weight links taking into account that density factor. Links that are surrounded by “more” text are probably more important to topic than links positioned in groups, but groups of links can signify we are on the hub page that can also be important to our focused crawler. Can we learn positions of interesting links for some topics? In any case, we believe, information about position and belonging to a certain area can help to infer if link is promising or not!

To note the final example, consider the problem of cheating search engines by inserting irrelevant keywords into an HTML source. This is widely used technique in order to raise the probability of indexing the page by search engine and representing it with higher rank among search results. While it is relatively easy to detect and reject false keywords where their foreground color is same as background color, there is no way to detect keywords of regular color but covered with images. If coordinates of objects in a page representation are known, then search engines could filter false keywords hidden by other objects and users would get better answers on their queries!

All the previously mentioned issues motivated us to define new representation of a page extracted from an HTML source, which include visual information, and to show how it can be utilized in recognition of common areas in a web page. However, we were additionally encouraged to do this work when discovering the fact that users expect from web designers to put certain objects at predefined areas on the browser screen [6]. Figure 1 shows where users expect to find internal and external links. As we shall see, this fact (and of course corresponding real situation on the Web) will help us to define heuristics for recognition of common page areas (menus, header, footer and “center” of the page).

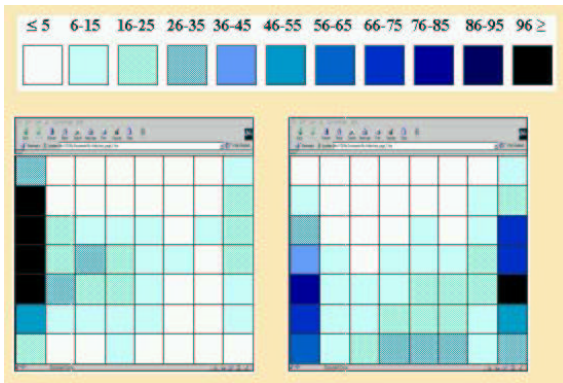


Figure 1: User expectance in percents concerning positions of internal (left) and external (right) links in the browser window [6].

The outline of the paper is as follows: In Section 2 we define the *M-Tree* format of a page used to render the page on the virtual screen, i.e. to obtain coordinates for every HTML object. Section 3 describes heuristics for recognition of header, footer, left and right menu, and “center” of the page. In Section 4, experimental results in a recognition process on a predefined dataset are shown. In section 5 we present some results concerning Web page classification using previously extracted visual information. Finally, conclusion and remarks about the future work are given in Section 6.

2. Extracting visual information from an HTML source

We define a virtual screen (VS) that defines a coordinate system for specifying the positions of HTML objects (in further text - objects) inside Web pages (in further text - pages). The VS is a half strip with a predefined width and an infinite height both measured in pixels. The VS is set to correspond to the page display area in a maximized browser window on a standard monitor with resolution of 1024x768 pixels. Of course one can set any desirable resolution. Width of the VS is set to be 1000 because when vertical scroll bars from browser are removed, that quantity is usually left for rendering the page. Obviously pages are of different length and so theoretically height can be infinite. Top left corner of the VS represents the origin of the VS coordinate system.

Now, the process of the visual information extraction will be described. Main operations applied in the extraction process are shown in Figure 2. In the first step a page is parsed using an HTML parser that extracts two different types of elements – tags and data. Tag elements (*TE*) are delimited with $\langle \rangle$ while data elements (*DE*) are contained between two consecutive tags. Each *TE* includes name of the corresponding tag and a list of attribute-value pairs. *DE* is represented as a list of tokens, which taken all together form the data string between consecutive tags. Separators between tokens are white space characters and are omitted from the *DE* list of tokens. *DE* contains empty list if no token is placed between two consecutive tags (i.e. input stream is $\dots \rangle W \langle \dots$, *W*-white space character or none). Parser skips $\langle \text{SCRIPT} \rangle$ and $\langle /-\dots \rangle$ tags.

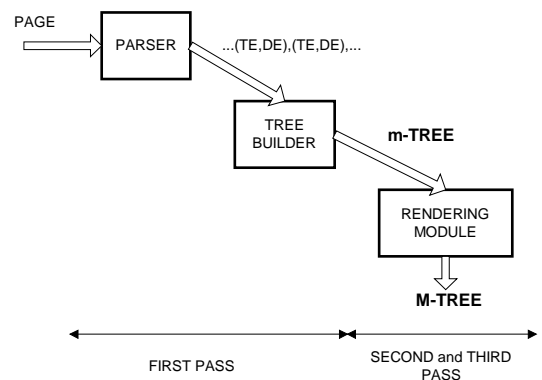


Figure 2: Constructing the *M-Tree* (main steps).

In the second step, as soon as $\langle TE, DE \rangle$ pair is extracted from the input HTML stream, it is injected into the tree builder. Tree builder applies stack machine and a set of predefined rules to build the tree that represents the HTML structure of the page. The output of this component we named *m-Tree*. There are many papers that describe construction of the parsing tree of

an HTML page [7][8]. In our approach a technique is adopted which constructs the tree in one single pass through the given page, i.e. parsing and building the *m-Tree* is done in a single pass. Rules are used to properly nest *TEs* into the hierarchy according to the HTML 4.01 specification [9]. Additional efforts were made to design tree builder that will be immune on bad HTML source. Now *m-Tree* (in further text *mT*) will be defined.

Definition 1: *mT* is directed n-ary tree defined with a set of nodes N and a set of edges E with following characteristics:

$$1. N = N_{desc} \cup N_{cont} \cup N_{data}$$

where:

- N_{desc} (description nodes) is a set of nodes, which correspond to *TEs* of the following HTML tags: $\{<TITLE>, <META>\}$

- N_{cont} (container nodes) is a set of nodes, which correspond to *TEs* of the following HTML tags: $\{<TABLE>, <CAPTION>, <TH>, <TD>, <TR>, <P>, <CENTER>, <DIV>, <BLOCKQUOTE>, <ADDRESS>, <PRE>, <H1>, <H2>, <H3>, <H4>, <H5>, <H6>, , , , <MENU>, <DIR>, <DL>, <DT>, <DD>, <A>, ,
, <HR>\}$

- N_{data} (data nodes) is a set of nodes, which correspond to *DEs*.

Each $n \in N$ has following attributes: *name* equals the name of the corresponding tag except for N_{data} nodes where *name* = "TEXT", *attval* is a list of attribute-value pairs extracted from the corresponding tag and can be null (i.e. nodes from N_{data} have this attribute set to null). Additionally, each N_{data} node has four more attributes: *value*, *fsize*, *emph*, and *align*. First contains tokens from the corresponding *DE*, second describes font size of these tokens, third carries information whether tokens belong to the scope of validity of one or more of the following HTML tags: $\{, <I>, <U>, , , <SMALL>, <BIG>\}$. The last one describes the alignment of the text (left, right or centered) In further text if n corresponds to tag X we write $n_{<X>}$ (n has attribute *name* = X).

2. Root of *mT*, $n_{ROOT} \in N_{cont}$ represents a page as a whole and its *name* is set to "ROOT" while *attval* contains only one pair (*URL* : *source url of the page itself*).

$$3. E = \{(n_x, n_y) \mid n_x, n_y \in N\}.$$

There can be only the following types of edges:

$$a) (n_{ROOT}, n_{desc}), n_{desc} \in N_{desc}$$

- b) $(n_{cont1}, n_{cont2}), n_{cont1} \in N_{cont} \setminus \{n_{}\}, n_{cont2} \in N_{cont} \setminus \{n_{ROOT}\}$ iff n_{cont2} belongs to the context of n_{cont1} according to the nesting rules of the HTML 4.01 specification
- c) $(n_{cont}, n_{data}), n_{cont} \in N_{cont} \setminus \{n_{}\}, n_{data} \in N_{data}$ iff n_{data} belongs to the context of n_{cont}

◆

From definition 1 it is clear that image and text nodes can be only leafs in an *mT*. Figure 3 shows a possible example of a simple page and its corresponding *mT*.

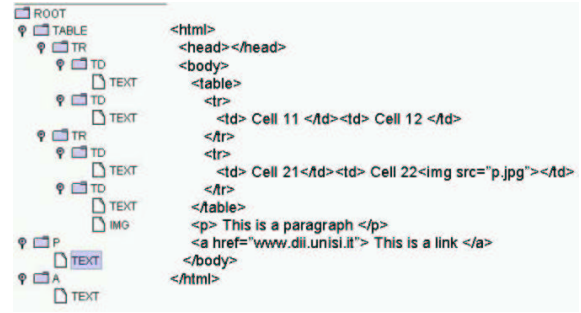


Figure 3: HTML source (right) and related *mT* (left)

After *mT* is obtained from the input page and when context of every object of interest is known, it is possible to apply algorithm for coordinate calculation. In fact, it is nearly the same algorithm that every browser does when rendering the page. Coordinates of objects are calculated in the third step using the rendering module (see Figure 2) and constructed *mT* as its input. We did not find any specific algorithm for page rendering except some recommendations from W3C [9] and so it was necessary to design our own. We decided to imitate the visual behavior of the Internet Explorer because of the popularity of this product. It is clear how difficult it could be if all aspects of the rendering process would be taken into account. Hence some simplifications are made, which by our opinion do not influence significantly on the final task – recognition of common areas in a page. The simplifications are as follows:

- 1) Rendering module (RM) calculates only coordinates for nodes in *mT*, i.e. HTML tags out of definition 1 are skipped.
- 2) RM does not support layered HTML documents.
- 3) RM does not support frames.
- 4) RM does not support style sheets.

Rendering module produces final, desirable representation of a page – *M-Tree* (in further text *MT*). *MT* extends the concept of *mT* by incorporating coordinates for each $n \in N \setminus N_{desc}$.

Definition 2: MT is the extension of mT with following characteristics:

1. For $\forall n \in N \setminus N_{desc}$, there are two additional attributes: X and Y . These are arrays which contain x and y coordinates of the corresponding object polygon on the VS.

2. If $n \in N_{cont} \setminus \{n_{<A>}\}$ then X and Y have dimension 4 and it is assumed that object represented with n occupies rectangle area on the VS. Margins of this n 's rectangle are:

- Down margin is equal to the up margin of the left neighbor node if exists. If n does not have a left neighbor or $n = n_{<TD>}$ then down margin is equal to down margin of n 's immediate parent. If $n = n_{ROOT}$ then down margin is the x -axes of the VS coordinate system.

- Up margin is equal to the up margin of the rightmost leaf node in the sub tree in which n is the root node.

- Left margin is equal to the left margin of n 's immediate parent, shifted to the right for the correction factor. This factor depends on the name of the node (i.e. if $name = LI$ this factor is set to 5 times current font width because of indentation of list items). If $n = n_{<TD>}$ and n has a left neighbor then left margin is equal to right margin of n 's left neighbor. If $n = n_{ROOT}$ then left margin is the y -axes of the VS coordinate system.

- Right margin is equal to the right margin of n 's immediate parent. If $n = n_{<TABLE>}$ or $n = n_{<TD>}$ then right margin is set to correspond to table/cell width.

3. If $n \in N_{data}$ or $n = n_{<A>}$ then X and Y can have dimension from 4 to 8 depending on the area on the VS occupied by the corresponding text/link (see Figure 4). Coordinates are calculated assuming the number of characters contained in the *value* attribute and current font width. Text flow is restricted to the right margin of the parent node and then new line is started. Heights of lines are determined by current font height.

◆

Previous definition covers most aspects of the rendering process, but not all because of the complexity of the process. For example if the page contains tables then RM implements modified auto-layout algorithm [9] for calculating table/column/cell widths. So when $n_{<TABLE>}$ is encountered, RM makes one more pass from that node down the mT to calculate cell/column/table widths. Hence the first pass is dedicated to table width calculations, and in the second pass RM calculates final coordinates for nodes that belong to the observed sub tree. If there are other $n_{<TABLE>}$ nodes down on the path (nesting of tables in the page) the process of calculating widths is recursively performed, but with no additional

passes. Before resolving a table, artificial cells (nodes) are inserted in order to simplify calculus in cases where cell spanning is present (*colspan* and *rowspan* attributes in a $<TD>$).

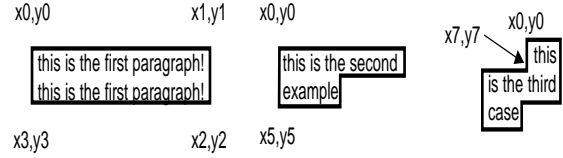


Figure 4: Some of the possible $TEXT$ polygons

Let us consider the complexity of the MT extraction process. First and second step (extracting $<TE,DE>$ pairs and building the mT) are performed in a single pass through the page. Hence the complexity so far is $O(s)$, where s represents the size of the file. In the third step RM transforms mT into MT while passing through mT and calculating coordinates for every non-descriptive node. If mT does not contain nodes that represent table TEs (tables in a page) then one single pass in the third step is needed and complexity remains linear. If the page contains tables then in the worst case RM performs one pass more. Hence the complexity of the RM phase is $O(2s)$ and the resulting complexity of the MT extraction process is $O(3s)$ which is satisfactory for most applications.

3. Defining heuristics for recognition of common areas of interest

Given the MT of a page and assuming the common web design patterns, it is possible to define a set of heuristics for recognition of standard areas in a page such as menu or footer. First, areas of the interest are listed to be header (H), footer (F), left menu (LM), right menu (RM), and center of the page (C). At present there are no exact definitions in the open literature for these page areas (one can think of these areas as groups of objects). Therefore we adopted intuitive definitions of these areas, which rely exclusively upon VS coordinates of logical groups of objects in a page. It is helpful to understand these groups of objects as frequently found areas in pages regardless of a page topic. They are tightly related to the presentational concept of a page. Naturally, heuristics based on visual information are used to recognize them, instead of exact algorithms. After careful examination of many different pages on the Web, we restricted the areas in which H , F , LM , RM , and C can be found. Before we describe what is recognized to be H , F , LM , RM , and C , we will introduce the specific partition of a page as it is shown in figure 5.

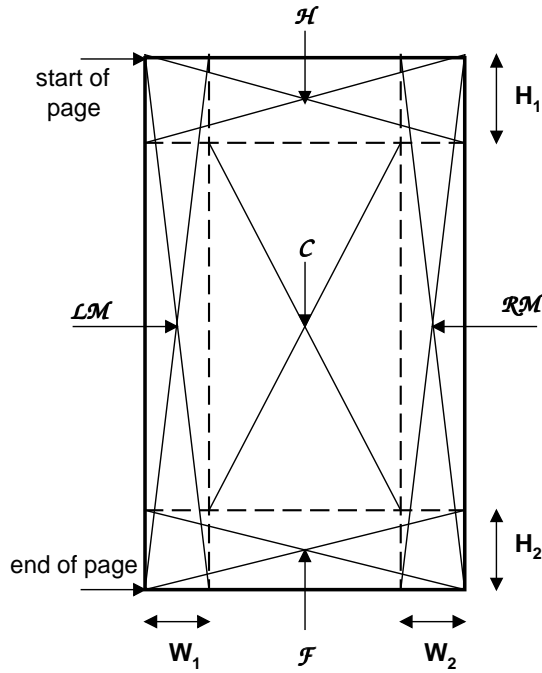


Figure 5: Position of areas of interest in a page

We set $W_1 = W_2$ to be 30% of the page width in pixels determined by rightmost margin among nodes from MT . W_1 and W_2 define LM and RM respectively which are locations where LM and RM can be exclusively found. We set $H_1 = 200$ pixels and $H_2 = 150$ pixels. H_1 and H_2 define H and F respectively which are locations where H and F can be exclusively found. Of course one can set different values, but initial experiments showed previous values are appropriate (see Section 4). Now we define following heuristics:

Heuristic 1: H consists of all nodes from MT that satisfy one or more of the following conditions:

1. Subtree S of MT with its root r_S belongs to H iff r_S is of type $n_{<TABLE>}$ and $n_{<TABLE>}$ completely belongs to H (i.e. upper bound of a table is less than or equal to H_1).

2. Subtree S of MT with its root r_S belongs to H iff upper bound of r_S is less than or equal to m and r_S does not belong to sub trees found in 1. Number m is the maximum upper bound of all $n_{<TABLE>}$ nodes found in 1. ♦

Heuristic 2: LM consists of all nodes from MT that are not contained in H and satisfy one or more of the following conditions:

1. Subtree S of MT with its root r_S belongs to LM iff r_S is of type $n_{<TABLE>}$ and $n_{<TABLE>}$ completely belongs to LM (i.e. right bound of a table is less than or equal to W_1).

2. Subtree S of MT with its root r_S belongs to LM iff r_S is of type $n_{<TD>}$, and $n_{<TD>}$ completely belongs to LM , and $n_{<TABLE>}$ to which this $n_{<TD>}$ belongs has lower bound less than or equal to H_1 , and upper bound greater than or equal to H_2 .

♦

Heuristic 3: RM consists of all nodes from MT that are not contained in H , LM and satisfy one or more of the following conditions:

(Similar as heuristic 2 except RM and W_2 instead of LM and W_1)

♦

Heuristic 4: F consists of all nodes from MT that are not contained in H , LM , RM , and satisfy one or more of the following conditions:

1. Subtree S of MT with its root r_S belongs to F iff r_S is of type $n_{<TABLE>}$ and $n_{<TABLE>}$ completely belongs to F (i.e. down bound of a table is greater than or equal to H_2).

2. Subtree S of MT with its root r_S belongs to F iff lower bound of r_S is greater than or equal to m and r_S does not belong to sub trees found in 1. Number m is the maximum lower bound of all $n_{<TABLE>}$ nodes found in 1.

3. Let $n \in \{n_{
}, n_{<HR>}\}$ or n is in the scope of the central text alignment. Further, assume n is the lowest of all nodes in MT completely contained in F . Sub tree S of MT with its root r_S belongs to F iff lower bound of r_S is greater than or equal to upper bound of n , and r_S does not belong to sub trees found in 1 and 2.

♦

Heuristic 5: C consists of all nodes from MT that are not in H , LM , RM , and F . ♦

From previous definitions of heuristics one can realize the importance of the $<TABLE>$ tag and its related tags $<TR>$ and $<TD>$. These tags are commonly used ($\approx 88\%$) for purposes not originally intended by inventors of HTML [10]. Web designers usually organize layout of the page and alignment of objects by including a lot of tables in a page. Therefore every table cell often represents a smallest amount of logically grouped information, visually presented to the user in a browser window (in our case on the VS). The same stands for tables that often group menu objects, footers, search and input forms, and other common page objects. Realization of the previous heuristics is done in at most 2 additional passes through the given MT . Hence the resulting complexity of the whole recognition process is nearly $O(5s)$, and that allows us to apply it in different applications mentioned in Section 1.

4. Experimental results in a recognition process

An experiment is performed to show how efficient can be recognition process using only visual information given through *MT*. The setup of the experiment was as follows:

Step 1: Construct the dataset *D* that contains sufficient number of different pages from different sites.

Step 2: Walk through *D* manually and label areas that can be considered as *H*, *F*, *LM*, *RM*, and *C*.

Step 3: Perform automatic extraction of *MT* for each page in *D*. Perform automatic recognition of areas of interest using defined heuristics on *MT*. Make automatically labeled new dataset *D₁* from each previously processed *MT*.

Step 4: Walk through *D₁* manually and estimate how well areas are recognized using manually labeled *D* as a reference point.

Step 1 is conducted by downloading nearly 16000 pages from the open source directory www.dmoz.org as a starting point for our crawler. We downloaded nearly 1000 files from the first level of each root category. *D* is constructed from the downloaded set by randomly choosing 515 files, uniformly distributed among categories and also in size. Two persons performed step 2 once. Second person was a kind of control and ultimate judge for labeling. Step 3 is performed using *Siena Tree* tool that includes *MT* builder and logic for applying recognition heuristics. *Siena Tree* is written in Java 1.3 and can be used to visualize objects of interest from a web page. For example one can see in a scrolling window where are the paragraphs and line breaks placed (*<P>* and *
*). Also one can enter any sequence of HTML tags to obtain picture (visualization) of their positions. Again, two persons in step 4 make judgment of recognizer performance by entering into each labeled file and comparing automatic labels with hand made labels from step 2. After step 4 we got the results shown in Table 1.

In order to discuss results from Table 1, notions of “*bad*” or “*good*” in recognition process have to be clarified. If area *X* exists but it is not labeled at all, or if *X* does not exist but something is labeled as *X*, then mark “*not recognized*” is evidenced. If less than 50% of objects that belong to *X* are labeled, or if some objects out of *X* are labeled too, then mark “*bad*” is evidenced. Mark “*good*” is evidenced if more than 50% but less than 90% of objects from *X* are labeled and no objects out of *X* are labeled. Mark “*excellent*” is evidenced if more than 90% of objects from *X* and no objects out of

X are labeled. Verification process was very tedious and it lasted one week!

	Header	Footer	Left M	Right M	Overall
Not recognized	25	13	6	5	3
Bad	16	17	15	14	24
Good	10	15	3	2	50
Excellent	49	55	76	79	23

Table 1: Success in recognition process (in %). Shaded rows represent successful recognition.

We stress that mark “*bad*” is given in cases where something is wrongly recognized. That is because we intend to use *Siena Tree* to filter the noise for the text classification purposes. Therefore if some text from the center of the page is wrongly removed we could lose important information. Also, recognition of *C* is, according to heuristic 5, complementary to recognition of other areas. So we did not include it in performance measurements. Results from column “overall” are obtained by introducing the total score *S* for the page *P* as a sum of all marks for recognition of all areas of interest. If $X \in \{H, F, LM, RM\}$ is “*not recognized*” then corresponding mark is 0. Marks “*bad*”, “*good*”, and “*excellent*” are mapped into 1, 2, and 3 respectively. Now, if $S=12$ we assume recognition process for particular file (page) is “*excellent*”. Similar “*good*” stands for $8 \leq S < 12$, “*bad*” stands for $4 \leq S < 8$, and “*not recognized*” stands for $S < 4$. Analyzing pages that perform as “*bad*” or “*not recognized*” we found that in nearly 20% the *MT* was not quite correct but the *mT* was correct i.e. rendering process was not good enough. Typical error is that portions of a page are internally good rendered but they are scrambled as a whole. For the rest of 80% of “*not recognized*” and “*bad*” recognized pages we suppose defined heuristics are not sufficient enough. Finally we selected values for margins H_1 , H_2 , W_1 , and W_2 according to statistics from [6]. In further research other values have to be considered as well.

5. Page classification using visual information

The rendering module provides an enhanced document representation, which can be used whenever the traditional bag-of-words representation cannot capture the complex structure of a Web page (i.e. page ranking, crawling, document clustering and classification). In particular, we have performed some document categorization experiments using the rich representation provided by the rendering module (*MT* representation).

At the time of writing, there is not a dataset of Web pages, which has been commonly accepted as a standard

reference for classification tasks. Thus, we have decided to create our own. After extracting all the URLs provided by the first 5 levels of the DMOZ topic taxonomy, we selected 14 topics at the first level of the hierarchy (we rejected topic “*Regional*” which features many non-English documents). Each URL has been associated to the class (topic) from which it has been discovered. Finally, all classes have been randomly pruned, keeping only 1000 URLs for each class. Using a Web crawler, we downloaded all the documents associated to the URLs. Many links were broken (server down or pages not available anymore), thus only about 10.000 pages could be effectively retrieved (an average of 668 pages for each class). These pages have been used to create the dataset.

Such dataset¹ can be easily replicated, enlarged and updated (the continuous changing of Web format and styles does not allow to employ a frozen dataset since after few months it would be not representative of the real documents that can be found on the Internet).

5.1. Naive Bayes Classifier

The Naive Bayes classifier [11] is the simplest instance of a probabilistic classifier. The output $p(c/d)$ of a probabilistic classifier is the probability that the pattern d belongs to class c (posterior probability).

The Naive Bayes classifier assumes that text data comes from a set of parametric models (each single model is associated to a class). Training data are used to estimate the unknown model parameters. During the operative phase, the classifier computes (for each model) the probability $p(d/c)$ expressing the probability that the document is generated using the model. The Bayes theorem allows the inversion of the generative model and the computation of the posterior probabilities (probability that the model generated the pattern). The final classification is performed selecting the model yielding the maximum posterior probability.

In spite of its simplicity, the Naive Bayes classifier is almost as accurate as state-of-the-art learning algorithms for text categorization tasks [12]. The Naive Bayes classifier is the most used classifier in many different Web applications such as focus crawling, recommending systems, etc. For all these reasons, we have selected such classifier to measure the accuracy improvement provided by taking into account visual information.

5.2. Classification results

The dataset was split into training and a test set of equal size. First, a Naive Bayes classifier was trained on all the words in the documents. Such classifier is usually constructed when not considering visual information and it provides a baseline to validate the effectiveness of

the proposed data representation. In order to classify a page taking into account its visual appearance, each page from the training (test) set was processed, extracting the bag-of-words representation of its six basic constituents: header, footer, left menu, right menu, center and title and meta-tags (see Figure 6). Then, we created six Naive Bayes classifiers where the i -th classifier was trained using the bag-of-words representations of the i -th constituents of the documents (i.e. the third classifier has been trained on the words belonging to the left menu of the documents).

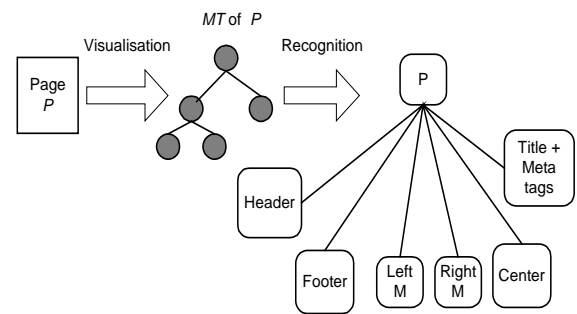


Figure 6: Page representation used as an input for six Naive Bayes classifiers

When classifying a document, the i -th classifier assigns a score to the document equal to $p_i(c/d)$. Mixing the scores of each single classifier performs the final decision. The mixture is performed assigning to the i -th classifier a weight w_i taking into account the expected relevance of the information stored into a specific part of the page:

$$p(c/d) = \sum_i w_i * p_i(c/d)$$

In particular, after some tuning we have assigned the following weights to each classifier: header 0.1, footer 0.01, left menu 0.05, right menu 0.04, center 0.5, title and meta-tags 0.3. Table 2 shows the classification results of the proposed method. Taking into account the visual appearance of the page that is provided by *MT*, we achieved an improvement of more than 10% in the classification accuracy.

6. Conclusion

This paper describes a possible representation for a web page in which objects are placed into well-defined tree hierarchy according to where they belong in an HTML structure of a page. We named this representation *M-Tree*. Further, each object (node from *M-Tree*) carries information about its position in a browser window. This visual information enables us to define heuristics for recognition of common areas such as header, footer, left and right menus, and center of the page. The crucial difficulty was to develop sufficiently

¹ The data set can be downloaded from <http://nautilus.dii.unisi.it/download/webDataset.tar.gz>

good rendering algorithm i.e. to imitate behavior of popular user agents such as Internet Explorer.

	Correct (no visual)	Correct (visual)	Total
Arts	105	176	324
Business	123	224	316
Computers	129	182	319
Games	141	212	339
Health	271	284	380
Home	284	269	348
Kids & Teens	81	171	343
News	198	218	336
Recreation	161	161	338
Reference	232	180	320
Science	121	163	335
Shopping	126	203	304
Society	194	180	337
Sports	145	222	341
Total	2311 (49%)	2845 (61%)	4680 (100%)

Table 2. Comparison of the classification accuracy of a Naive Bayes classifier when taking into account a bag-of-words representation of the page versus a mixture of Naive Bayes classifiers taking into account the visual appearance of documents. Information carried by the visual appearance increases classification accuracy of more than 10%.

We concluded from analyzed pages that HTML source was often far away from the proposed standard and it posed additional problems in rendering process. After applying some techniques for the error recovery in construction of the parsing tree and introducing some rendering simplifications (we do not deal with frames, layers and style sheets) we defined recognition heuristics based only on visual information. We could have included other types of information into recognition process, but we wanted to observe percentage of successfully recognized areas based only on page layout structure and common design patterns. The overall success in recognizing targeted areas yields 73%. From Table 1 one can see that menus are either recognized or not. On the other hand recognition of header and footer is more complex and heuristics other than just visual have to be considered. In further research we plan to improve rendering process and recognition heuristics.

Some preliminary results have shown that spatial information is important to classify Web documents. Classification accuracy of a Naive Bayes classifier was increased of more than 10%, when taking into account the visual information. In particular, we constructed a mixture of classifiers each one trained to recognize words appearing in a specific portion of the page. In the future, we plan to use Neural Networks to find the optimal weights of our mixture of classifiers. We hope our system will also improve focus crawling strategies

[4] by estimating importance of the link based on its position and neighborhood. We believe that our visual page representation can find its application in many other areas related to search engines, information retrieval and data mining from the Web.

7. References

- [1] Quinlan, J.R., "Induction of decision trees", *Machine Learning*, 1986, pp. 81-106.
- [2] Salton, G., McGill, M.J., *An Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.
- [3] Chakrabarti S., van den Berg M., Dom B., "Focused crawling: A new approach to topic-specific web resource discovery", *Proceedings of the 8th Int. World Wide Web Conference*, Toronto, Canada, 1999.
- [4] Diligenti M., Coetzee F., Lawrence S., Giles C., Gori M., "Focused crawling using context graphs", *Proceedings of the 26th Int. Conf. On Very Large Databases*, Cairo, Egypt, 2000.
- [5] Rennie J., McCallum A., "Using reinforcement learning to spider the web efficiently", *Proceedings of the Int. Conf. On Machine Learning*, Bled, Slovenia, 1999.
- [6] Bernard L.M., "Criteria for optimal web design (designing for usability)", <http://psychology.wichita.edu/optimalweb/position.htm>, 2001
- [7] Embley D.W., Jiang Y.S., Ng Y.K., "Record-Boundary Discovery in Web Documents", *Proceedings of SIGMOD*, Philadelphia, USA, 1999.
- [8] Lim S. J., Ng Y. K., "Extracting Structures of HTML Documents Using a High-Level Stack Machine", *Proceedings of the 12th International Conference on Information Networking ICOIN*, Tokyo, Japan, 1998
- [9] World Wide Web Consortium (W3C), "HTML 4.01 Specification", <http://www.w3c.org/TR/html401/>, December 1999.
- [10] James F., "Representing Structured Information in Audio Interfaces: A Framework for Selecting Audio Marking Techniques to Represent Document Structures", Ph.D. thesis, Stanford University, available online at <http://www-pcd.stanford.edu/frankie/thesis/>, 2001.
- [11] Mitchell T., "Machine Learning", McGraw Hill, 1997.
- [12] Sebastiani F., "Machine learning in automated text categorization", *ACM Computing Surveys*, 34(1), pp. 1-47