Identifying Table Boundaries in Digital Documents via Sparse Line Detection

Ying Liu, Prasenjit Mitra, C. Lee Giles
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA, USA, 16802
yliu@ist.psu.edu, pmitra@ist.psu.edu, giles@ist.psu.edu

ABSTRACT

Most prior work on information extraction has focused on extracting information from text in digital documents. However, often, the most important information being reported in an article is presented in tabular form in a digital document. If the data reported in tables can be extracted and stored in a database, the data can be queried and joined with other data using database management systems. In order to prepare the data source for table search, accurately detecting the table boundary plays a crucial role for the later table structure decomposition. Table boundary detection and content extraction is a challenging problem because tabular formats are not standardized across all documents. In this paper, we propose a simple but effective preprocessing method to improve the table boundary detection performance by considering the *sparse-line* property of table rows. Our method easily simplifies the table boundary detection problem into the sparse line analysis problem with much less noise. We design eight line label types and apply two machine learning techniques, Conditional Random Field (CRF) and Support Vector Machines (SVM), on the table boundary detection field. The experimental results not only compare the performances between the machine learning methods and the heuristical-based method, but also demonstrate the effectiveness of the sparse line analysis in the table boundary detection.

Categories and Subject Descriptors

H.3 [**INFORMATION STORAGE AND RETRIEVAL**];
H.3.7 [**Digital Libraries**]

General Terms

Algorithms, Performance

Keywords

Table Boundary Detection, Sparse Line Property, Table Labeling, Conditional Random Field, Support Vector Machine, Table Data Collection

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.
Copyright 2008 ACM 978-1-59593-991-3/08/10 ...$5.00.

1. INTRODUCTION

Table, as a specific document component, is widely used in web pages, scientific documents, financial reports, etc. Researchers always use tables to concisely display the latest experimental results or statistical financial data in a condensed fashion. Other researchers, for example, who are conducting the empirical studies in the same topic, can quickly obtain valuable insights via examining these tables. Along with the rapid expansion of the Internet, tables become a valuable information source in the information retrieval field. Based on the increasing demands to unlock the information in tables, more applications appear in the table-related fields, e.g., the table search [12]. Although approaches on table analysis are diverse, they share two same analyzing steps: table boundary detection and table structure decomposition. For the further table content storage and sharing (e.g., the table data extraction and the table search), locating the table boundary in a document is the first and crucial step.

Different from most table detection works, which are the pre-defined layout based and the rule-based methods, we apply machine learning techniques on the table boundary detection field in this paper. Pre-defined layout based algorithms usually work well for one domain, but are difficult to extend. For the rule-based methods, the performance is always heavily affected by the quality of the rules. When the testing data set is large enough, it is difficult to determine the "good" values for thresholds. Machine learning methods are good choices to deal with such problems. Wang et al. [23] applied the decision tree and Support Vector Machine (SVM) techniques to classify the web tables into genuine tables and non-genuine tables. However, their work starts with the identified tables without any detail about how to detect the table boundary in a document page. David et al. [19] compared the experimental results of extracting the tables from plain-text government statistical reports using Conditional Random Fields (CRF) and Hidden Markov Models (HMM) respectively. Unfortunately, many adopted line labels (e.g., separator) in [19] are too specific to be applicable in other document medium (e.g., HTML, PDF).

In general, table boundary detection problem can be transformed into the problem of identifying the table lines, which constitute the table boundaries. By the observation of tables with diverse layouts from different documents, we identify that all the table lines share an important property: *majority lines belonging to the table areas are sparse in terms of the text density.* Existing *filter-out* based table-line discovering methods identify the table lines from the entire set of lines of a document according to certain rules, which in turn results in low *recall*. However, for applications such as table search [12], *recall* is more important than *precision* because once the false negative table lines are removed, it is difficult to retrieve them back. However, the false positive rate can be easily lowered in later table structure decomposition step. In this paper, we propose a novel but effective method to quickly locate the boundary of a table by taking advantage of the aforementioned property. We also propose an *exclusive* based method for identifying table lines, which generates high recall and saves substantial effort to analyze the noisy lines. To our best knowledge, there are no proposed works that compare the machine learning based methods with the rule-based methods on the table boundary detection. In this paper, we apply two machine learning methods (CRF and SVM) on the table boundary detection Furthermore, we elaborate the feature selection, analyze the factor effects of different features, and compare the performance of CRF/SVM approaches with our proposed rule-based method.

Instead of extracting tables from the HTML documents [23] or the plain-text documents [19], we focus on Portable Document Format (PDF). PDF is a widely used document format in digital libraries because it can preserve the appearance of the original document. Although a good number of researches have been done to discover the document layout by converting the PDFs to other types of files (e.g., image, html, text) in the past two decades, automatically identifying the document logical structures information (e.g., words, text lines, paragraphs, etc) and extracting the document components (e.g., figures, tables, mathematical formulas, etc) as well as the content [2] are still a challenging problem. The major reasons are as follows: 1) the structural information is not explicitly marked up because of the un-tagged nature of PDF format; 2) the text sequences are often messily generated by the existing PDF-to-text tools; 3) new noises can be generated by some necessary tools (e.g., OCR), if converting the PDFs into other media (e.g., image).

The rest of the paper is organized as follows. Section 2 reviews several relevant studies in table boundary detection area and the applied machine learning methods in this field. Section 3 introduces the sparse-line property of the table lines. Section 4 describes in detail the sparse line detection and the noise line removing using the conditional random field and support vector machine (SVM) techniques. We elaborate the label types and the feature sets. Section 5 explains the line construction before the line labeling. Section 6 explains how to locate the table boundary based on the labeled lines as well as the table keywords. The detailed experimental results are displayed in Section 7. We conclude our paper with plans for future work in Section 8.

2. RELATED WORKS

2.1 Related works on Table Detection

Researchers in the automatic table extraction field largely focus on analyzing the table structure in a specific document media. Chen et al. [3] used heuristic rules and cell similarities to identify tables. They tested their table detection algorithm on 918 tables from airline information web pages and achieved an F-measure of 86.50%. Penn et al. [18] proposed a set of rules for identifying genuinely tabular information and news links in HTML documents. They tested their algorithm on 75 web site front-pages and achieved an F-measure of 88.05%. Yoshida et al. proposed a method to integrate WWW tables according to the category of objects presented in each table [27]. Their data set contains 35,232 table tags gathered from the web. They estimated their algorithm parameters using all table data and then evaluated algorithm accuracy on 175 of the tables. The average F-measure reported in their paper is 82.65%.

Zanibbi [28] provides a survey with detailed description of each method. All the methods can be divided into three categories: pre-defined layout based [22], heuristics based [17, 6, 9, 8], and statistical based. Pre-defined layout based algorithms usually work well for one domain, but is difficult to extend. Heuristics based methods need a complex post-processing and the performance relies largely on the choice of features and the quality of training data. Most approaches described so far utilize purely geometric features (e.g. pixel distribution, line-art, white streams) to determine the logical structure of the table, and different document mediums require different process methodologies: OCR [19], X-Y cut [4], tag classification and keyword searching [10][3][24] etc. In the past two decades, a good number of researches have been done to discover the document layout by converting the PDFs to image files. However, the image analysis step can introduce noise (e.g., some text may not be recognized or some images may not be correctly recognized). In addition,because of the limited information in the bitmap images, most of them only work on some specific document types with minimal object overlap: e.g., business letters, technical journals, and newspapers. Some researchers combine the traditional layout analysis on images with low-level content extracted from the PDF file. Even if the version 6 of PDF allows a user to create a file containing structure information, most of them do not contain such information.

Chao et al. [2] reported their work on extract the layout and content from PDF documents. Hadjar et al. have developed a tool for extracting the structures from PDF documents. They believe that, to discover the logical components of a document, all/most of the page objects need to be analyzed such as text objects, image objects, path objects, etc, which are listed by PDF document content stream. However, the object overlapping problem happens frequently. If all the objects are analyzed, more effort needs to be spent to firstly segment these objects from each other. In addition, even such objects/structures are identified, they are still too high level to fulfill many special goals, e.g., detecting the tables, figures, mathematical formulas, footnotes, references, etc. Instead of converting the PDF documents into other types of media (e.g., image or HTML) and then applying the existing techniques, we process PDF documents directly from the text level.

2.2 Related works on table analysis with machine learning approaches

Several machine learning approaches are applied in the table analysis field, e.g., decision tree [20], Naive Bayes classifier [29], Support Vector Machine (SVM) [1], Conditional random fields (CRF) [11]etc. Hurst mentioned in [5] that a Naive Bayes classifier algorithm produced adequate results but no detailed algorithm and experimental information was provided. Wang et. al. tried both the decision tree classifier and SVM to classify each given table entity as either *genuine* or *non-genuine* table based on features from layout, content type, and word group perspectives. Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating discrete-valued functions that is robust to noisy data. Comparing with our work, they started with the detected tables and all features are related to the table itself (e.g., the number of columns). How to detect these tables, the key problem of our paper, is missing in their work.

Conditional random fields (CRFs) is initially introduced by Lafferty et al. [11] in 2001 as a framework for building probabilistic models to segment and label sequence data. After the birth, CRF is applied in bio-informatics, computational linguistics and speech recognition fields. Conditional Random Fields (CRF) have been shown to be useful in part-of-speech tagging [11], shallow parsing [21], named entity recognition for newswire data [15], as well as table detection [19]. To the best of our knowledge, Pinto et al. [19] did the most related work as we did. Comparing with our work, the difference spans the following areas: 1) they extract table from a more specific document type – plain-text government statistical reports; 2) because of the specific document nature, they adopted several special labels and corresponding features (e.g., BLANKLINE label and SEPARATOR features), which are not applicable for other document types; 3) their features focus on white space, text, and separator instead of of the coordinate features, which are important for the table structure decomposition; 4) although they claimed that their paper concentrated on locating the table and identifying the row positions and types, they did provide the detail about the table locating. In order to improve the performance table data extraction, we zoom in the table boundary detection problem and elaborate the feature selection in our paper. Moreover, we consider the coordinate features, which not only play a crucial role in the table boundary field, but also are unavoidable in the later cell segmentation phase. Different from most CRF applications, the input data is a document line instead of a word.

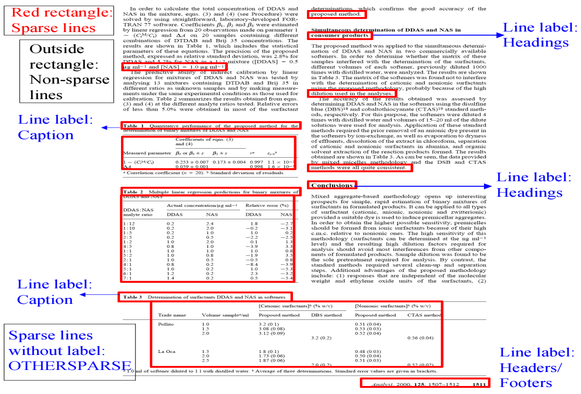


Figure 1: The sparse lines in a PDF page

3. THE SPARSE-LINE PROPERTY OF TABLES

Tables present structural data and relational information in a two-dimensional format and in a condensed fashion. Scientific researchers always use tables to concisely display their latest experimental results or statistical data. Other researchers can quickly obtain valuable insights by examining and citing tables. Tables have become an important information source for information retrieval. The demand for locating such information (table search) is increasing. To successfully get the table data from a PDF document, detecting the boundary of the table is a crucial phase. Based on the observation, we notice that different lines in the same document page have different widths, text densities, and the sizes of the internal spaces between words. A document page contains at least one column. Many journals/conferences require two (e.g., ACM and IEEE templates) or three even four columns. In a document, some lines have the same length as the width of the document column, some are longer (e.g., cross over multiple document columns) or shorter (e.g., the heading "1. INTRODUCTION" in our paper) than a column. From the internal space perspective, the majority of the lines contain normal space sizes between adjacent words while some lines have large spaces. In this paper, we define *sparse line* as follows.

Definition 1. ***Sparse Line:*** *A document line is a sparse line if any of the following condition is satisfied: 1). The minimum space gap between a pair of consecutive words within the line is larger than a threshold* sg. *2). The length of the line is much shorter than a threshold* ll*;*

Since the majority of the lines in a document belong to the non-sparse category, separating the document lines into sparse/non-sparse categories according to the text internal space/density and then getting rid of the non-sparse category become a fruitful preprocessing step for the table boundary detection. Such a method has two advantages: 1) the sparse lines cover nearly the entire table content lines; 2) Narrowing down the table boundary to the sparse lines at the early stage can save substantial time and effort to analyze noise lines.

There are tables whose cells can cross over multiple table columns. In order to collect all such cells, method proposed in [26] sets up constraints on the number of such long cells within a table boundary. However, determining a reasonable value is difficult. For example, if the value is set up too tight, part of a table could be missed out. If the value is loose, noise lines will be included into the table boundary. Unlike the approach in [26], our method treats the long cells as non-sparse lines and remove temporally. To decide whether a sparse line should be included into the same table boundary, we only need to check the vertical space gaps between this sparse line and its previous neighbor sparse line. Once we merge these two sparse lines into the same table boundary, the previously temporally removed long lines (if exists any) between those two sparse lines should be retrieved back.

Different definitions of the "much shorter than" may generate different sparse line labeling results. We define it as the half of the document column width. We show a snapshot of a PDF document page in Figure 1 as an example. We highlight the sparse lines in red rectangles. Apparently, the table body content lines are labeled as sparse lines. Ten sparse lines are not located within the table boundary: two heading lines, one footer line, three caption lines, and four short lines that are the last line in a paragraph. We label them as sparse lines because they satisfy the second condition. Since such short-length lines also happen in some table rows with only one filled cell, we consider them as sparse lines to avoid missing out the potential table lines. Such noise non-table sparse lines are very few because they usually only exist at the headings or the last line of a paragraph. In addition, the short length restriction also reduce the frequency. We can easily get rid of them based on the coordinate information later.

4. MACHINE LEARNING TECHNIQUES

4.1 Support Vector Machines

SVM [1] is a binary classification method which finds an optimal separating hyperplane $x : wx + b = 0$ to maximize the margin between two classes of training samples, which is the distance between the plus-plane $x : wx + b = 1$ and the minus-plane $x : wx + b = -1$. Thus, for separable noiseless data, maximizing the margin equals minimizing the objective function $||w||^2$ subject to $\forall i, wyi(x_i + b) \geq 1$. In the noiseless case, only the so-called support vectors, vectors closest to the optimal separating hyperplane, are useful to determine the optimal separating hyperplane. Unlike classification methods where minimizing loss functions on wrongly classified samples are affected seriously by imbalanced data, the decision hyperplane in SVM is not affected much. However, for inseparable noisy data, SVM minimizes the objective function: $||w||^2 + C \sum_{i=1}^{n} \varepsilon_i$ subject to $\forall i, wy_i(x_i + b) \geq 1 - \varepsilon_i$, and $\varepsilon_i \geq 0$, where ε_i is the slack variable, which measures the degree of misclassification of a sample x_i. This noisy objective function has included a loss function that is affected by imbalanced data.

In order to increase the importance of recall in SVM, a cut-off classification threshold value $t < 0$ should be selected. In methods with outputs of class probability $[0, 1]$, then a threshold value $t < 0.5$ should be chosen. As noted before, for noiseless data, SVM is stable, but for noisy data, SVM is affected much by imbalanced support vectors. In our work, the latter approach is applied for SVM, i.e., when $t < 0$, recall is to be improved but precision decreases. When $t > 0$, a reverse change is expected.

4.2 Conditional Random Fields

Conditional Random Fields (CRFs) are undirected statistical graphical models, which are well suited to sequence analysis. The primary advantage of CRFs over Hidden Markov Models (HMM) is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference. Additionally, CRFs avoid the label bias problem, a weakness exhibited by Maximum Entropy Markov Models (MEMMs) and other conditional Markov models based on directed graphical models. Let $\mathbf{o} =< o_1, o_2, ..., o_n >$ be an sequence of observed input data sequence, for example in our case as a sequence of input lines of text in a PDF document page. Let $\mathbf{S}$ be a set of states in a finite state machine, each corresponding to a label $l \in L$ (e.g., sparse line, non-sparse line, heading line, etc.) Let $s =< s_1, s_2, ..., s_n >$ be the sequence of states in $\mathbf{S}$ that correspond to the labels assigned to the lines in the input sequence $\mathbf{o}$. Linear-chain CRFs define the conditional probability of a state sequence given an input sequence to be:

$$P(s|\mathbf{o}) = \frac{1}{Z_o} exp(\sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_j f_j(s_{i-1}, s_i, \mathbf{o}, i)) \qquad (1)$$

where Z_o is a normalization factor of all state sequences, the sum of the "scores" of all possible state sequences. $f_j(s_{i-1}, s_i, \mathbf{o}, i)$ is one arbitrary feature function of m functions that describes a feature over its arguments, and λ_j is a learned weight for each such feature function.

$$Z_o = \sum_{s \in S} exp(\sum_{i=1}^{n} \sum_{j} \lambda_j f_j(s_{i-1}, s_i, \mathbf{o}, i)) \qquad (2)$$

Intuitively, the learned feature weight λ_j for each feature f_j should be positive for features that are correlated with the target label, negative for features that anti-correlated with the label, and near zero for relatively uninformative features. These weights set to maximize the conditional log likelihood of labeled sequences in a training set $D = < o, \mathbf{l} >_{(1)}, ..., < o, \mathbf{l} >_{(n)}$:

$$LL(D) = \sum_{i=1}^{n} log(P(\mathbf{l}_{(i)}|\mathbf{o}_{(i)}) - \sum_{j=1}^{m} \frac{\lambda_j^2}{2\sigma^2} \qquad (3)$$

When the training state sequence are fully labeled and unambiguous, the objective function is convex, thus the model is guaranteed to find the optimal weight settings in terms of $LL(D)$. Once these settings are found, the labeling for a new, unlabeled sequence can be done using a modified Viterbi algorithm.

We use a weight parameter θ to boost features corresponding to the true class during the testing process. Similar to the classification threshold t in SVM, θ can tune the trade-off between recall and precision, and may be able to improve the overall performance, since the probability of the true class increases. During the testing process, the sequence of labels s is determined by maximizing the probability model $P(s|\mathbf{o}) = \frac{1}{Z_o} exp(\sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_j f_j(s_{i-1}, s_i, \mathbf{o}, i, \theta_s))$, where $f_j(s_{i-1}, s_i, \mathbf{o}, i, \theta_s) = \sum_{i=1}^{|} o|\theta_{s_i} t_j(s_{i-1}, s_i, \mathbf{o}, i)$, θ_s is a vector with $\theta_{s_i} = \theta$ when $s_i = true$, or $\theta_{s_i} = 1$ when $s_i = false$, and λ_j is the parameters learned while training.

4.3 Line Labels

Different from the traditional table boundary detection works, we use an exclusive method to label all the potential table lines. Figure 2 shows the inclusion-relation of the line types in a document page. The size of each block does not represent the ratio of a line type in the page. Each line type corresponds to a label in the machine learning methods.

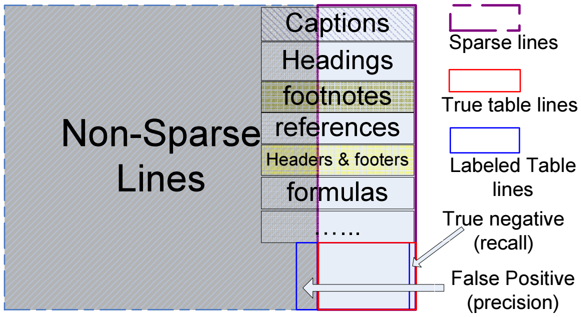


Figure 2: Composition of a PDF page with line types

We design a set of labels by examining a large number of lines in scientific PDF documents. Each line will be initially labeled as either *SPARSE* or *NONSPARSE*. A line labeled as *NONSPARSE* satisfies neither of the conditions in Section 3. *NONSPARSE* lines usually cover the following document components: document title, abstract, paragraphs, etc. *SPARSE* lines cover other specific document components entirely/partially: tables, mathematical formulas, texts in figures, short headings, affiliations, document headers and footers, and references, etc.

Even though sparse lines cover almost all table lines, a few *non-table* lines mingle in. Removing these noise lines can facilitate the table boundary detection efficiently. Therefore, for the labeled *SPARSE* lines, we label them as the following six categories: *CAPTIONSPARSE*, *HEADINGSPARSE*, *FOOTNOTESPARSE*, *REFERENCESPARSE*, *HEADERFOOTERSPARSE*, and *OTHERSPARSE*. *CAPTIONSPARSE* refers to a line that is the first line of a table caption or a figure caption. *HEADINGSPARSE* marks short document headings. Usually the lines labeled with the *HEADINGSPARSE* or the *CAPTIONSPARSE*, or *FOOTNOTESPARSE* only satisfy the second condition mentioned in Section 3. To label a line with these labels, additional features should be considered. For *HEADINGSPARSE* lines, the font size and type are the key features. For *CAPTIONSPARSE* lines, we should examine whether the line starts with the defined keywords or not (e.g., Table or Figure). For *FOOTNOTESPARSE* lines, the specific starting symbol is the most important factor. To identify the *HEADERFOOTERSPARSE* lines, checking the Y-axis coordinate is key. Although a large part of lines with these labels also exist in non-sparse line group, we can easily zoom in the table boundary into the last category *OTHERSPARSE* by removing such lines from sparse line set with this method.

4.4 Feature sets

Wise choice of features is always vital to the final results. The feature based statistical model CRFs reduce the problems to finding an appropriate feature set. This section outlines the main features used in these experiments. Overall, our features can be classified into three categories: the *orthographic features*, the *lexical* features, and the *document layout features*. Instead of the features about white space and separators in [19], we emphasize the layout features.

4.4.1 Orthographic features

Most related works treat the vocabulary as the simplest and most obvious feature set. Such features define how these input data appear (e.g., capitalization etc), based on regular expressions as well as prefixes and suffixes. Because the line layout is much more important than their appearance for our line labeling problem, we do not have to consider so many orthographic features as they did. Our orthographic features include: *InitialCaptical, AllCaptical, FontSize, FontType, BoldOrNot, HasDot, HasDigital, AllDigital*, etc.

4.4.2 Lexical features

The lexical features includes: *TableKwdBeginning, FigureKwdBeginning, ReferenceKwdBeginning, AbstractKwdBeginning, SpecialCharBeginning, DigitalBeginning, SuperscriptBeginning, SubscriptBeginning, LineItself.*

4.4.3 Layout features

Our crucial features come from the layout perspective. The layout features include: *LineNumFromDocTop, LineNumToDocBottom, NumOfTextPieces, LineWidth, CharacterDensity, LargestSpaceInLine, LeftX, rightX, MiddleX, DisToPrevLine, DisToNextLine*. Table 1 lists the detailed description for every layout feature.

Table 1: The main document layout features in our experiment

Document Layout Features	Description
$LineNumFromDocTop$	Index number from the page top
$LineNumToDocBottom$	Index number to the page bottom
$NumOfTextPieces$	Number of text pieces
$LineWidth$	Width of the line
$CharacterDensity$	$LineWidth$/Number of characters
$LargestSpaceInLine$	Largest space within the line
$LeftX$	X-axis value of leftmost line end
$EndX$	X-axis value of rightmost line end
$MiddleX$	X-axis value of middle point
$TheDisToPrevLine$	Vertical gap to the previous line
$TheDisToNextLine$	Vertical gap to the next line

4.4.4 Conjunction features

So far all the previous features are described over a single predicate. In order to capture relationships that a linear combination of features cannot capture, we look at the conjunction of features. CRFs provide the function to determine the label of a line by taking into account information from another line. In our work, we set the window size of features as *-1, 0, 1* to conjunct the current line with the previous line and the following file. In order to avoid the overflowed memory and exacerbated overfiting generated by the all possible conjunctions and only generate those feature conjunctions with significant improvement functions, we turn to feature induction as described in [14]. We start with no feature and choose new features interactively. In each iteration, we evaluate some sets of candidates using the Gaussian prior, and add the best ones into the model.

5. LINE CONSTRUCTION IN PDFS

Different from most CRF applications, the unit of our problem is a document line, instead of a single word. Before classifying the document lines, we have to construct the lines first. To construct the document lines, we deal with the PDF source file character by character as well as the related glyph information through analyzing the *text operators*[1]. Adobe's Acrobat word-finder provides the coordinate of the four corners of the quad(s) of the word. The PDFlib Text Extraction Toolkit (TET) also provides the function to extract the text in the different levels (character, word, line, paragraph, etc.). However, it only provides the content instead of other style information in all the levels except the character level. If we want to do some further work, content itself is usually not enough. We have to calculate the corresponding coordinates for the higher levels by merging the characters. Similar to Xpdf library, we adopt a bottom-up approach to reconstruct these characters into words then lines with the aid of their position information and saves the results. To convert characters into words then lines, we adopt some heuristics based on the distance between characters/words. For each document page, we construct lines according to their internal word relative position information and width. Within a same word, different characters have the same font properties. However, within a same line, font diversity may exist among different words (e.g., the superscript, the subscript, or mathematical symbols). The main unique place of our method is that we only analyze the

[1] PDF Reference Fifth Edition, Version 1.6

coordinate information. Font information, the frequently adopted parameter, is not used in our method as the rule to decide whether merge the next word into the same line or not. Therefore, the font information does not affect the performance of the sparse line detection.

Table 2: The parameter thresholds we adopted for word reconstruction

P	Definition
α	the vertical distance between two top Y-axis values: $alpha = Y_{i+1} - Y_i$
β	the vertical distance between two bottom Y-axis values: $beta = Y'_{i+1} - Y'_i$
γ	the horizontal distance between these two characters: $\gamma = X_{i+1} - X'_i$
δ	the vertical distance of two characters
θ	the maximal width of the space with a word
η	the maximum vertical distance between two characters in a same line

Formally, we define a document as a set of pages D $= \cup_{k=1}^{n}(P_k)$, where n is the total page number. Each page P_k, can be denoted as an aggregation of characters $C \in \{Character\}$. c_i and c_{i+1} are a pair adjacent (no other character exists between them) characters. Initially, we get the coordinate of the first character c_0 in a document page. All the characters in C share a common set of attributes $\{([X, X'], [Y, Y'], W, H, F, T)\}$, where $[X, Y]$ is the pair of coordinators of the upper-left corner of the character while $[X', Y']$ is the coordinates of the bottom-right corner of the character. The original point of the X-Y axes is the left-bottom corner of a document page. W/H denotes the width/height of the component, F is the font size, and T is the text. Figure 3 shows the coordinates of an example character. not use the font size because in many journals and archives, the font information (the font type and the font size) is not so standard as we imaged. Considering such unreliable information will incur more error to the final results.

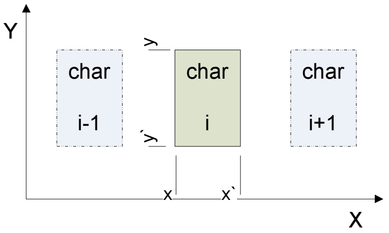


Figure 3: The coordinates of a character in a PDF document page.

Since the character C is the fundamental component of a document, other components can be constructed recursively from it. For example, a document *page* P_k can be denoted as an aggregation of *words* $W = \{w_j | w_j = ([X_{w_j}, Y_{w_j}]), ([X'_{w_j} Y'_{w_j}]), W_{w_j}, H_{w_j}, F_{w_j}, T_{w_j}\}$. A document word w_j is equal to $\cup_{i=1}^{m} c_i$, where m is the total number of characters in the word w_j. Figure 4 enumerates all the relative positions of a pair of adjacent characters c_i and c_{i+1}. Their coordinates are $([X_i, X'_i], [Y_i, Y'_i])$ and $([X_{i+1}, X'_{i+1}], [Y_{i+1}, Y'_{i+1}])$ respectively. For the word reconstruction, we define several parameters and thresholds, which are listed in Table 2:

Figure 4 (a) presents a common character pair in the same line. $Y_{i+1} = Y_i (\alpha = 0)$ and $Y'_{i+1} = Y'_i (\beta = 0)$. If γ is smaller than a given threshold θ, the second character c_{i+1} can merge with c_i into a same word. Otherwise, we treat c_i as the last character of the current word and c_{i+1} as the starting character of a new word.

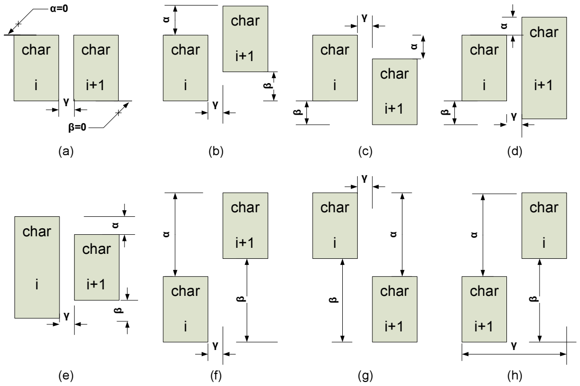


Figure 4: The coordinates of the example cases of the character pairs

Figure 4 (b) – (e) display several examples of the same-line character neighbors with partial vertical overlaps. Superscript is a typical case of Figure 4 (b) while subscript is a typical case of Figure 4 (c). Figure 4 (d) and (e) show the font size changing within a document line. All these character pairs are also same-line characters. Every case has to satisfy some fixed conditions as follows: $Y_{i+1} \geq Y_i \geq Y'_{i+1} \geq Y'_i$ (Figure 4 (b)), $Y_i \geq Y_{i+1} \geq Y'_i \geq Y'_{i+1}$ (Figure 4 (c)), $Y_{i+1} \geq Y_i \geq Y'_i \geq Y'_{i+1}$ (Figure 4 (d)), and $Y_i \geq Y_{i+1} \geq Y'_{i+1} \geq Y'_i$ (Figure 4 (e)). For these cases, we decide whether c_i and c_{i+1} go into the same word or not, by comparing γ with the same threshold θ;

To analyze the character pairs in Figure 4 (f) – (h), we introduce another threshold η: *the maximum vertical distance between two characters in a same document line*. In $4(f)$, the fixed constraint is $\delta = (Y'_{i+1} - Y_i) > 0$. In Figure $4(g)$ and (h), the fixed constraint is $\delta = (Y'_i - Y_{i+1}) > 0$. If $\delta > \eta$, C_i and C_{i+1} will belong to different lines. Otherwise, we treat them as the character neighbors in a same line and decide whether they go to the a same word. Starting a new document column in a page is a typical example with large γ for case f, and starting the next line is a typical example with large but minus γ for case h. Using the Table 1 in Figure1 as the example, we show the merged words in Figure 5. Each red rectangle refers an independent word.

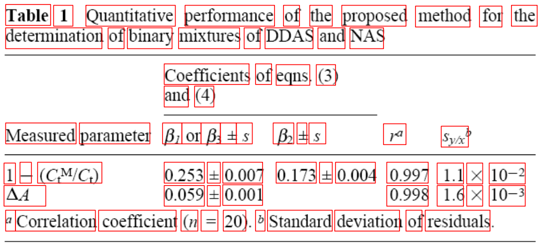


Table 1 Quantitative performance of the proposed method for the determination of binary mixtures of DDAS and NAS

Measured parameter	Coefficients of eqns. (3) and (4) β_1 or $\beta_3 \pm s$	 $\beta_2 \pm s$	r^a	$s_{y/x}{}^b$
$1 - (C_t{}^M/C_t)$	0.253 ± 0.007	0.173 ± 0.004	0.997	1.1×10^{-2}
ΔA	0.059 ± 0.001		0.998	1.6×10^{-3}

[a] Correlation coefficient (n = 20). [b] Standard deviation of residuals.

Figure 5: The merged words in a table after the character → word phase

Now a document page p_k can be denoted as an aggregation of words W. Similar to the characters, we can also treat words as rectangle objects in a document page. The coordinate nature of characters in the previous section is also applicable to the words. For a pair of word neighbors, w_i and w_{i+1}, the possible relative locations are same of the cases listed in Figure 4. Using the concept in Section 5.1, we should treat the *word* here as the *character* there and treat the *text piece* here as the *word* there. We believe that in the non-sparse lines, all the words can be merged into one piece. The parameters and the thresholds in Table 2 can be reused with only the value resets of γ and θ. Still using the Table 1 in Figure1 as the example, we show the merged lines in Figure 6.

After the combination, we check the number of text pieces in each line along the Y-axis sequence, if the number is larger than one, we label this line as the sparse line. If the number is one but it satisfy the first condition in Section 3, we also treat it as a sparse line. Still using the Table 1 in Figure1 as the example, we show the merged lines in Figure 6. For all the eight lines, the number of text pieces are *1, 1, 1, 1, 5, 5, 4,* and *1* respectively. We treat line 5, 6, and 7 are sparse lines because they contain more than one text piece. We also treat the line 3 and 4 as sparse lines because of the small width.

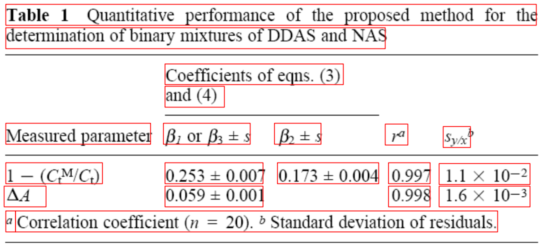


Table 1 Quantitative performance of the proposed method for the determination of binary mixtures of DDAS and NAS

	Coefficients of eqns. (3) and (4)			
Measured parameter	β_1 or $\beta_3 \pm s$	$\beta_2 \pm s$	r^a	$s_{y/x}{}^b$
$1-(C_t^M/C_t)$	0.253 ± 0.007	0.173 ± 0.004	0.997	1.1×10^{-2}
ΔA	0.059 ± 0.001		0.998	1.6×10^{-3}

[a] Correlation coefficient (n = 20). [b] Standard deviation of residuals.

Figure 6: The merged lines in a table

6. DETECTING THE TABLE BOUNDARY BASED ON THE KEYWORDS

After the sparse line detection and noisy line removal, we can easily detect the table boundary by combining the lines labeled as *OTHERSPASE* with the table keywords. Here we define the main table content rows as the table boundary, which does not have to include the table caption and the footnote. In order to enhance the performance of the table starting location detection, we consider the keyword information. We can directly detect the table boundary by detecting the tabular structure within the sparse line areas.

In our method, we define a keyword list, which lists all the possible starting keywords of table captions, such as "Table, TABLE, Form, FORM," etc. Most tables have one of these keywords in their captions. If more than one tables are displayed together, the keyword is very useful to separate the tables from one another. Once we detect a line (not only the sparse line) starting with a keyword, we treat it as a table caption candidate. Then we check other lines that are located around the caption and merge them into a sparse area according to the vertical distances between adjacent lines. Such sparse-line areas are the detected table boundary. The vertical distance is the key feature to filter out most remaining noise lines. Because the texts within the detected table boundary will be analyzed carefully in the later table structure decomposition phase, we treat *recall* more important than *precision* here. Once we locate the table boundary, we check this area and try to retrieve the long table lines that are labeled as non-sparse lines to improve the *recall.*

7. EXPERIMENTS AND RESULTS

In this section, we demonstrate the experimental results of evaluating our table boundary detection with two machine learning methods. Our experiments can be divided into four parts: the performance evaluation of different methods, different feature settings, different datasets, and different parameter settings.

7.1 Data Set

We focus on tables in PDF scientific documents. Although Wang [25] tried to build a general table ground truth database, he focused on the web tables. No benchmark dataset exists in PDF table analysis field. In our work, we directly analyze PDF documents instead of converting them to HTML or Image file.

Instead of analyzing tables from a specific domain, we aim to collect tables as much different varieties as possible from digital libraries. The collection of this paper comes from diverse journals and proceedings in three sources: chemical scientific digital libraries (Royal Chemistry Society[2]), Citeseer[3], and archeology[4] in chemistry, computer science and archeology fields. The size of each PDF repository we collected exceeds $100,000$, $10,000$ and $8,000$ respectively in terms of scientific papers. All the documents span the years 1950 to 2008. From these documents, we randomly choose 300 pages with and without tables for our experiments as the training set. Among these pages, we refer 100 pages from the chemistry field as the dataset H, 100 pages from the computer science field as the dataset S, and 100 come from the archeology field as the dataset A. The total number of the lines in three datasets are 10177, 13151, and 9741 respectively. For every document line, we manually identify it with a label defined in section 5.3. In order to get an accurate and robust evaluation on the table boundary detection performance, we adopt a hold-out method by randomly dividing the dataset into five parts and in each round we train four of the five parts and tested on the remaining one part. The final overall performance comes from the combined five results. In our experiment, we use the java-implemented, first-order CRF implementation – Mallet – to train two versions of the CRF with binary features and the actual values. For SVM, we adopt SVM light [7].

We divide the table boundary detection problem into four main sub-problems as follows: 1) Construct the lines in a document page; 2) Remove all the non-sparse lines from the line set; 3) Remove all the noisy sparse lines; 4) Label table lines by considering the keywords. In this section, we check the performance of each step and analyze the impact effect of different feature set.

7.2 Text Extraction from PDFs

PDF document content stream lists each PDF document as a sequence of pages, which in turn can be recursively decomposed into a series of components, such as text, graphics, and images. The corresponding objects to those com-

[2] http://www.rsc.org/
[3] http://citeseer.ist.psu.edu/
[4] http://www.saa.org/publications/AmAntiq/AmAntiq.html

ponents are text objects, image objects, path objects, etc. Most of the existing works to discover the logical components of a document focus on analyzing most if not all of the page objects. For example, regrouping all the objects to form the image is a traditional task for document analysis system. However, the object overlapping problem happens frequently and the researchers have to make more effort to segment objects from each other first. Even when such objects or structures are identified, they are still too high level to fulfill specific goals, including our table boundary detection. For most table related applications (e.g., table data extraction and table search), the majority research interests are focused on the text (the table content), instead of the borderlines. We classify the tables into two categories according to the content type: the *text table* and the *image table*. Text tables refer to those tables that all parts are composed of texts. Image tables refer to those tables that are image themselves or contain images in some cells. All three tables in Figure 1 are text tables. By random examining thousands of the PDFs in the computer science, chemistry, biology, and archeology fields, we notice that more than 92% table contents consist of pure texts while only few tables contain images. Because most tables are composed of texts, the text extracting tools, which only provide the very low level information (characters, words, coordinates, etc) without the structure information, is enough for our goal. Many PDF converters are available off the shelf (Xpdf, PDF2TEXT, PDFBOX, Text Extracting tool, PDF-TEXTSTREAM, etc) to extract texts from documents. The information obtained with the help of these tools can be divided into two categories: the *text content* and the *text style*. The text content refers to the text strings; The text style includes the corresponding text attributes: the font, the size, line spacing and color, etc; We tried all the tools and TET has the best performance on text extraction. The text streams extracted from PDF files can correspond to various objects: a character, a partial word, a complete word, a line, etc. In addition, the order of these text streams does not always correspond to the reading order. A line reconstruction and a reading order resorting steps are necessary in order to correctly extract the text from a PDF file. The details of our text sequence resorting algorithm is beyond the topic of this paper and can be found in [13]. In this paper, after the line reconstruction and the sparse line detection, it is assumed that the text sequence is correct as a matter of course.

7.3 Performance of line construction

We evaluate the performance of our line construction in Section 5 based on 300 selected PDF pages. Given the number of total lines T, the number of constructed lines that do not have any error C. Usually an error line contains at least one of the following problems: 1) the constructed line includes some texts that should not belong to it; 2) the constructed line misses a part of the text; If the missed texts are included in the previous/next line, we do not count the error duplicately. Within the 33069 lines in the 300 PDF pages, we accurately constructed 99.057%(32757) lines. The main reason for the errors is the superscripts/subscripts in the documents with dense layouts.

7.4 Performance of sparse line detection

We perform a five-user study to evaluate the quality of the sparse line detection. Each user checks the detected sparse lines in 20 randomly selected PDF document pages in each dataset. The evaluation metrics are *precision* and *recall*. we define the recall as $\frac{ts}{ts+fn}$ and the precision as $\frac{ts}{ts+fp}$. ts refers to the labeled true positive sparse lines in a page. fn refers to the false negatives (they are sparse lines but we miss them). fp refers to the false positives (they are not sparse lines, but we label them as the sparse lines). Table 4 displays the results based on the 300 PDF pages.

Table 3: The performance evaluation of the sparse line detection

datasets	H	A	S
The Number of PDF pages	100	100	100
Recall of sparse line detection	99.82	99.37	99.52
Precision of sparse line detection	98.60	99.22	98.79

There are two goals in our method: 1) removing non-sparse lines as much as possible; 2) keeping true table lines in the sparse line set as much as possible. To achieve a more objective evaluation, we also check the performance of this step from the perspectives of these two goals. Some tables have long cells and very small spaces between the adjacent table columns because of the crowd layout. In order to keep these table lines (goal 2), we regulate thresholds by setting ll with a tolerate value and sp with a smaller value. The trade off is mislabeling some non-sparse lines as sparse lines. Because we have further steps to remove the noise from the sparse lines, including such non-sparse lines (low *precision*) is not a big problem.

Within the datasets H, A and S, 84.63% lines are labeled as non-sparse lines and can be easily removed as noise. Within the remaining sparse lines, which account for 15.37% in the whole dataset, almost half (44.23%) of them are real table lines and 95.35% table lines are included in the sparse line set. There are two reasons for the missed table lines: 1) we label some table lines as non-sparse lines because they contain long cross-column cells without large space gap. Such missed lines can be retrieved in section 6. 2) the text missing problem inherited from the text extraction tools. This deficiency falls outside the topic our paper.

7.5 Performance of Noise line removal

We use the same test data in section 7.3 to evaluate the performance of the noise line removal. The measurement methods are still *precision* and *recall*. Let *tl* be the real table lines that are kept in the sparse line set, *sp* be the latest size of the sparse line set after each noise removal, and *to* be the real table lines that are removed. We define the *precision* as $\frac{tl}{sp}$ and *precision* as $\frac{tl}{tl+to}$.

In Figure 7 (a), X-axis lists all noise type to be removed from the sparse line set as well as the postprocessing in Section 6. FP refers the beginning dataset – all lines in a page. RNS refers the non-sparse lines. RH refers the noisy heading lines. HF refers the noisy header and footnote lines. CAP is the noisy caption lines, REF is the noisy reference lines, and PP represents the postprocessing step. Along with the noise removing, the size of the sparse line dataset decreases and the *precision* of the table line labeling increase steadily. Non-sparse line removing and the postprocessing are two crucial steps for the table boundary detection problem. The results on three datasets are consistent without any remarkable difference. The *precision* value is improved from 8.59% to 59.72% on average after removing all noises. In addition, the further steps are much easier because of the dramatically reduced sparse line set. Although the results are not good enough, the remaining false positive table lines scatter the page and the large distance to the table caption is an important feature to identify them.

Figure 7 (b) shows the *recall* curves with the same experimental conditions. The initial *recall* values are 100% because no line is removed. Along with each step, the recall is decreasing because few true table lines are mislabeled and removed. Within three datasets, dataset S has the worst recall value because most computer science documents do not follow the standard template strictly and some true table lines are mislabeled.

7.6 Keyword Combination

After the noise removal in the previous section, the typical false positive table lines are the lines with short length. Such lines are usually located at the end of paragraphs, the last line of a table caption, or a short table footnote without special beginning symbol etc. Considering the distance features, most of the first type can be filtered out. For those missed true table lines, analyzing the location information of adjacent sparse line sections together with the table caption help us to retrieve them back. Based on the method in Section 6, the *precision* values is enhanced to 95.32% and the *precision* values is close to 98.34%.

7.7 Impact effects of feature sets

In order to compare the impact effect of different feature sets, we implement three set of experiments completed in the time allotted: one CRF model using only the orthographic features described in section 5.4.1, the second system adds the lexical feature set, and the third system uses all features. Every model is tested with all three datasets separately. The results are listed in Table 4. We use the results based on the rule-based method as the comparison baseline. The evaluation metrics are *precision*, *recall* and *F*-measure. Given the number of the correctly-labeled true table lines by each method A, the number of true positive table lines but overlooked B, and the number of true negative non-table lines that is misidentified as table lines C, the *Precision* is $\frac{A}{A+C}$, the *Recall* is $\frac{A}{A+B}$, and the *F-measure=(2*Recall*Precision)/(Recall+Precision).*

Table 4: Average accuracy of table boundary detection with different feature sets after all the noise line removal

feature sets, and datasets	Recall	Precision
CRF, Orthographic, H	42.18%	44.96%
CRF, Orthographic, S	41.66%	45.14%
CRF, Orthographic, A	40.89%	45.16%
CRF, Orthographic+Lexical, H	61.22%	61.66%
CRF, Orthographic+Lexical, S	59.30%	59.81%
CRF, Orthographic+Lexical, A	59.98%	60.58%
CRF, Orthographic+Lexical+Layout, H	**98.92%**	**96.28%**
CRF, Orthographic+Lexical+Layout, S	97.33%	96.49%
CRF, Orthographic+Lexical+Layout, A	98.76%	96.20%

It is not surprising to notice that within the multiple feature sets, the layout features play the most important impact effect on the final performance of the table line detection. For the rule-based method and CRF with less features, the *recall* values are always less than that of *precisions.* As the joining of more layout features, not only the overall table boundary detection performance is heavily increased, but also the recalls exceed the precisions, and satisfy the nature of the further table data search demand. Within the experiments with the same method and the same features, different training datasets have similar performances. Usually the dataset H has better performance comparing the datasets S and A because that the document layout and table structure in chemical papers are more standard than those in other two fields.

Table 5: Average accuracy of table boundary detection with different methods after all the noise line removal

Method and datasets	F-measure
Rule-based method, $H + S + A$	91.93%
CRF, H+S+A	**96.36%**
SVM linear, H+S+A	**94.38%**
Max Ent in [19]	88.7%
CRF Binary in [19]	91.2%
CRF Continuous in [19]	91.8%
C4.5 in [16]	$< 95\%$
Bp in [16]	$< 91\%$
Det in [16]	$< 70\%$

7.8 Impact effect of parameters

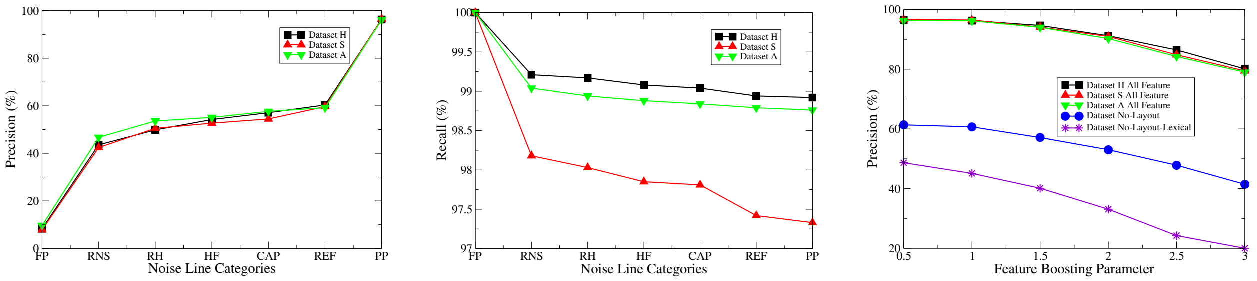
Figure 7(c) shows the effect of the feature boosting parameter θ for CRF. In the previous section, we use the default parameter setting: $\theta = 1.0$. In this section, we test different values: *0.5, 1.0, 1.5, 2.0, 2.5, 3.0.* If $\theta < 1.0$, the non-sparse lines get more preference. In each θ value, we compare the average precision results among different feature settings based on different datasets. We notice that no matter how we change the θ and datasets, more features generate better results than fewer features and the contribution of the *layout* feature is much higher than others. As the increasing of θ, the trend of precision is deceasing. The more features we consider, the more robust the results along the changing of θ.

7.9 Impact effect of different techniques

Moreover, we compare our results based on CRF and SVM methods with several main table boundary detection results published in other related works. The evaluation method is F-measure. For our CRF and SVM methods, we adopt all the features on all three datasets. Comparing with our previous rule-based method, our CRF experiments improve the performance by 54.90% and our SVM experiments improve the performance by 30.36%. Within all the published works, Ng et. al, achieved the best results with C4.5 method in [16]. Our work with CRF method enhance the F-measure by more than 27.2%.

8. CONCLUSIONS

In this paper, we propose a novel method to detect the table boundary. Because most tables are text-based, we claim that the text object of PDF provides enough information for table detection. Within the text object, we believe that the font size is not so reliable as other work stated. Based on the sparse-line nature of tables, we propose a fast but effective method to detect the table boundary by only processing the sparse lines in a document page. Processing the sparse lines solely can also improve the performance of the text sequence resorting problem. Combining different keywords,



(a) (b) (c)

Figure 7: The precision (a)/ recall (b) of table line labeling using CRF along the noise removing and postprocessing; (c) The precision of table line labeling using CRF with different θ

this method is applicable to detect other document components, e.g., references or headers/footers. Our next tasks include applying the machine learning methods on the table structure decomposition field and the table classification field. Moreover, extending our test data sets and building a general table benchmark database are also our future works.

9. REFERENCES

[1] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[2] H. Chao and J. Fan. Layout and content extraction for pdf documents. pages 213–224, 2004.

[3] S. T. H. Chen and J. Tsai. Mining tables from large scale html texts. In *In Proc. 18th Int'l Conf. Computational Liguistics, Saarbrucken, Germany*, 2000.

[4] J. Ha, R. Haralick, , and I. Philips. Recursive x-y cut using bounding boxes of connected components. In *In Proc. Third Int'l Conf. Document Analysis and Recognition*, pages 952–955, 1955.

[5] M. Hurst. Layout and language: Challenges for table understanding on the web, 2001.

[6] N. G. J. Shin. Table recognition and evaluation. In *In Proc. of the Class of 2005 Senior Conf., Computer Science Department, Swarthmore College*, pages 8–13, 2005.

[7] T. Joachims. Svm light. http://svmlight.joachims.org/.

[8] T. Kieninger and A. Dengel. Applying the t-rec table recognition system to the business letter domain. In *In Proc. of the 6th Int'l Conf. on Document Analysis and Recognition*, pages 518–522, September 2001.

[9] T. G. Kieninger. Table structure recognition based on robust block segmentation. In *In Proc. Document Recognition V, SPIE, volume 3305*, pages 22–32, January 1998.

[10] B. Krupl, M. Herzog, and W. Gatterbauer. Using visual cues for extraction of tabular data from arbitrary html documents. In *In Proc. of the 14th Int'l Conf. on World Wide Web*, pages 1000–1001, 2005.

[11] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th ICML*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[12] Y. Liu, K. Bai, P. Mitra, and C. L. Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *JCDL*, pages 91–100, 2007.

[13] Y. Liu, P. Mitra, and C. L. Giles. Improving the table boundary detection in pdfs by fixing the sequence error of the sparse lines. In *Technical report*, 2008.

[14] A. McCallum. Efficiently inducing features of conditional random fields. In *Nineteenth Conference on UAI*, 2003.

[15] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, 2003.

[16] H. Ng, C. Lim, and J. Koo. Learning to recognize tables in free text, 1999.

[17] H. Ng, C. Y. Lim, and J. T. Koo. Learning to recognize tables in free text. In *In Proc. of the 37th Annual Meeting of the Association of Computational Linguistics on Computational Linguistics*, pages 443–450, 1999.

[18] G. Penn, J. Hu, H. Luo, and R. McDonald. Flexible web document analy- sis for delivery to narrow-bandwidth devices, 2001.

[19] D. Pinto, A. McCallum, X. Wei, and W. Bruce. Table extraction using conditional random fields. In *In proceeding of Proceedings of the 26th ACM SIGIR, Toronto, Canada*, July 2003.

[20] S. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. In *SMC(21), No. 3, May 1991, pp. 660-674*.

[21] F. Sha and F. Pereira. Shallow parsing with conditional random fields, 2003.

[22] J. Shamilian, H. Baird, and T. Wood. A retargetable table reader. In *In Proc. of the 4th Int'l Conf. on Document Analysis and Recognition*, pages 158–163, 1997.

[23] J. Wang and J. Hu. A machine learning based approach for table detection on the web. In *WWW'02*, pages 242–250, Nov 2002.

[24] Y. Wang and J. Hu. Detecting tables in html documents. In *In Proc. of the 5th IAPR DAS, Princeton, NJ*, 2002.

[25] Y. Wang, I. Philips, and R. Haralick. Automatic table ground truth generation and a background-analysis-based table structure extraction method. In *In Proc. of the 6th Int'l Conference on Document Analysis and Recognition*, page 528, September 2001.

[26] B. Yildiz, K. Kaiser, and S. Miksch. pdf2table: A method to extract table information from pdf files. *IICAI05*, (Pune, India), 2005.

[27] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the world wide web, 2001.

[28] R. Zanibbi, D. Blostein, and J. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. In *Int'l J. Document Analysis and Recognition, Vol. 7, No.1*, pages 1–16, 2004.

[29] Z. Zheng. Naive bayesian classifier committees. In *European Conference on Machine Learning*, pages 196–207, 1998.