Table Extraction From PDF Documents

Shriram Shridharan
shrirams@cs.wisc.edu

Vidhya Govindaraju
vidhya@cs.wisc.edu

Abstract

A table is a very important component of a digital document which contains rich information. To extract the content from the tables, the tables should be identified and the structure be decomposed. This paper models the problem both as a sequence labeling problem and as a classification problem and compares the results from the two approaches. We use conditional random fields to predict the structure of a sequence, which in this case is detecting tables and decomposing the rows in a table. We use SVM and Logistic Regressor to classify the lines in a page as a table or a non-table line and the lines in a table as a header or data row.

1 Introduction

Information in documents in presented in various forms and one of the most convinient and compact form is a table. Information in tables is presented to improve the understanding for humans but is not easy for the computers to decode them. With the rise of interest in constructing knowledge bases and relation extraction for various domains, the builders often look into tables for more obvious relations. For e.g. lets consider the task of building a knowledge base for geologists. Information such as total organic carbon, sulfur content in rocks etc are presented in geology journal articles more often in the form of tables. Even in computer science research papers, the researchers often use tables and figures as a form to project the more important results. Often the information in tables is the only source of that information. These factors motivated us to work on table extraction which is the first step in constructing knowledge bases from tables.

We look into the problem of identifying and decomposing tables in text based PDF documents which is increasingly becoming a popular format to present information. Working with text-based PDF documents pose a lot of interesting challenges. Unlike HTML documents, the PDF documents are not tagged and unlike the plain DOC/TXT files, very often the text extracted from the PDF document is not perfectly ordered. We use pdftohtml[1] to extract information from PDF documents and the result is a plain text file with sentences in the document(quite often the order is not perfect) and the position of a sentence in the document.

Our intuition in table extraction led us to believe that it is a sequence-labeling problem where the tag of the current sentence depends on the previous sentence tag. This was obvious as if the previous sentence is a HEADER row, the current sentence can either be HEADER 2 or DATA. But with our feature set, we found that traditional classification algorithms perform better in finding tables and decomposing the contents.

The rest of the paper is organized as follows. In Section 2, we present the Related works in Table Extraction and Table Decomposition. In Section 3, we present a short background the the machine learning approaches that we used. In Sention 4, we present our solution to the problem. In Section 5, we present our empirical evaluations and in Section 6, we present our conclusions and future work.

2 Related Work

Most of the proposed approaches in table extraction can be classified into i) Predefined structure based approaches ii) Heuristic based approaches iii) Statistical and Machine Learning based approaches

Croft et al. [5] uses both the layout and domain specific features to extract tables from HTML documents. They model the problem of decomposing each row of a table as a sequence labeling problem and employ conditional random fields to tag cells in a table. However, their feature set relies heavily on the structure of tables and is domain dependent. They expect dashed lines and white space markers to delineate table rows and data. These information is not present when we consider all forms of tables such as tables in text and image-based PDF documents.

Yildiz et al.[6] base their work primarily on heuristics to locate a table in pdf documents. They use the out-

[1]pdftohtml [http://pdftohtml.sourceforge.net]

put of pdftohtml to do preprocessing and detect tables. The absolute position of text elements in a page is leveraged in finding tables. Though the advantage is that it is domain independent and can be used to find tables in image-based documents(using the output of a OCR), the approach uses a lot of heuristics that may not scale to the variations in the way a table is presented in text.

Liu et al. [3] build a search engine for tables called TableSeer. They design a novel heuristics based method for table detection called the page box-cutting method to segment a page into different boxes where each box has a high cohesion but low coupling with other boxes. Nevertheless, Liu et al. work primarily on scientific pdf documents and make a lot of assumptions that do not work well in other cases. Eg: They assume that the table caption is always on the top of the table and there is a fixed font size for all table information.

Liu et al. [4] focus on the table detection in PDF documents. They make use of the property that in most PDF documents, the table lines are sparse. They train a CRF and a linear SVM to detect tables and report a F-measure of 96.36% for CRF and 94.38% for linear SVM.

Fang et al. [1] use SVM, Logistic Regression and Random Forests to classify table header and data. They compared both the heuristic based solutions and machine learning techniques for table header detection and found that the random forest based approach outperform all other approaches.

Our work on table detection is also primarily based on [4]. However, we differ from [4] in that i) We pose the problem both as a sequence labeling and a classification problem where each line is tagged as either TABLE LINE or NON TABLE LINE, whereas [4] posed it only as a sequence labeling problem to label each line into either NONSPARSE or one of a several kind of SPARSE lines. ii) Also, our work involves a more sophisticated and complex pre and post processing algorithms than described in [4]. iii) And we empirically compare the performance of CRF and a SVM with a Gaussian kernel (of degree 5) instead of a linear SVM as done in [4]. Our later efforts in table decomposition is based on the observations of [1] and we employ both sequence labeling (using CRF) and classification techniques (using Linear SVM and Logistic Regressor) to classify a header row and a data row.

3 Background

3.1 Conditional Random Fields

Conditional Random Fields (CRF) are undirected graphical models primarily used for sequence labeling. CRF models the posterior probability of predicting a label sequence given an input sequence.

The posterior probability is given by

$$P(O|I) = \tfrac{1}{Z} \exp \sum_{i=1}^{n} \sum_{j=1}^{m} L_j F_j(O_{i-1}, O_i, I, i) \quad (1)$$

where n is the number of training instances
m is the number of features
O is the Output (label) sequence
I is the Input sequence
L_j is the weight of the feature
f_j is the feature given previous state, current state, Input sequence and the test instance
Z is the Normalisation factor (sum over all possible output label sequences)

$$\sum_{i=1}^{n} f_j(O_{i-1}, O_i, I, i) := F_j(I, O) \qquad (2)$$

Inference is done by a variation of the Vitterbi algorithm as specified in [2]. Training is to done by setting the weights to maximize the conditional log likelihood of labeled sequences in the training set.

4 Our Solution

4.1 Architecture of Table Extractor

Figure 1 shows the architecture of our table extraction system. As can be inferred from the figure, our table extraction system is primarily divided into 2 major components: Table Detection and Table Decomposition. The Table Detection component detects the boundaries of the tables in the PDF documents and passes the detected tables to the Table Decomposition component. The Table Decomposition component processes the detected tables and identifies header rows/columns and data rows. We describe each component in detail in the following sections.

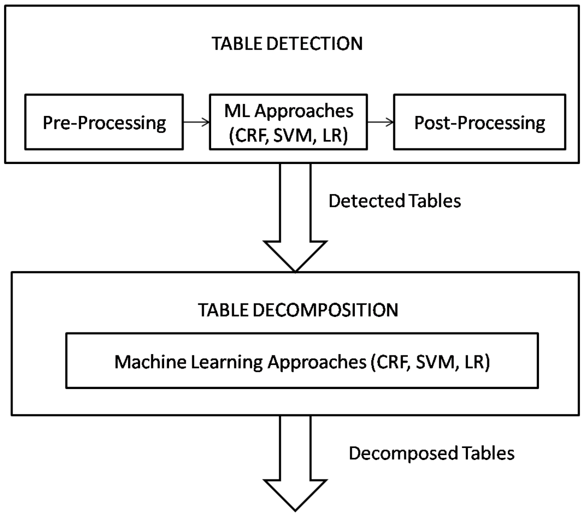

Figure 1: *Table Extraction - Architecture*

4.2 Pre-Processing

We use tohtml for converting the input text-based PDF document to an xml file. The xml output of pdftohtml consists of a set of page tags with each page tag consisting of a set of text tags and fontspec tags. The text tags consists of attributes like top, left, width, height and font. The fontspec tag consists of attributes like id, size, family and color. However,it does not handle correctly the column information (i.e. there is no column tags) or subscript/superscript information. In these cases, it splits the same sentence into multiple text tags and order of the text tags is not maintained. To overcome these limitations, we had to do preprocess the xml documents based on certain heuristics. The pseudocode of our preprocessing algorithm is as follows:

Algorithm 1 Preprocessing of input documents

```
for PAGE in input document do
    Combinesubscripts and superscripts based on top/height and left/width information
    Combine text pieces based on top information.
    Find columns in page based on sharp difference in height between previous and current tag
```

The result of the preprocessing stage is a list of all pages with each page containing the vertical columns in a page and the text in a page (with each text tag containing an extra 'textpieces' attribute).

4.3 Feature Sets

We used a wide variety of features as suggested in [4]. However, we also introduced some new features by inspecting the structure of different tables in PDF documents. The features can be broadly divided into orthographic, lexical, layout and other features. We used the same feature set to train different classifiers like CRF, SVM and Logistic Regressor except that only CRF had the previous and current tag information for each feature. We used forward selection on a set of features and selected those features that provided us better classification accuracy. We describe each feature in detail below.

4.3.1 Orthographic Features

1. Font Size - Same font size between previous and current line.
2. Begins With Captial Letter - First word begins with a capital letter in current line.
3. Percentage of numeric characters
4. Percentage of alphabetic characters

4.3.2 Lexical Features

1. Keyword Presence - Presence of 'Table' keyword followed by a number Eg: Table 1, Table 3.2

4.3.3 Layout Features

1. Textpieces - Using the 'textpieces' attribute added in the proprocessing stage to check if it is greater than a threshold
2. No. of words in line - If it is equal to 1. [This feature was useful because pdftohtml created a separate text tag with only one word for certain table lines]
3. Height Difference (Previous) - Difference in height between previous line and current line
4. Height Difference (Next) - Difference in height between next line and current line
5. Largest Space - If the largest space between any pair of consecutive words in the current line is greater than a threshold.

6. Same Space - If the largest space is equal to the smallest space between any pair of consecutive words in the current line.
7. No. of words in line - If it is greater than a threshold.
8. No. of words with the largest space difference - If the number of the words with largest space difference (Layout Feature (5)) is greater than a threshold.

4.3.4 Other Features (Only for CRF)

1. Previous and current tag are TABLELINE
2. Previous and current tag are NONTABLELINE

4.4 Machine Learning techniques employed

4.4.1 Conditional Random Fields

We implemented CRF using the algorithm specified by Charles Elkan in [2]. For the weight learning we use Stochastic Gradient Ascent with the weight update rule specified by the Collins Perceptron. However, we use weights averaged over a epoch instead of actual weights to prevent the magnitude of the weights from getting a value too high.

The weight update rule is as follows:

$$w_j := w_j + \lambda \tfrac{F_j(x,y) - F_j(x,y')}{N} \tag{3}$$

where w_j is the weight of feature j,
λ is the Learning rate,
$F_j(x,y)$ is the (Actual) Feature function for that feature j for label y,
$F_j(x,y')$ is the (Predicted) Feature function for that feature j for label y',
N is the Total number of training instances.

We also used a decayed learning rate for quicker convergence of weight learning. The decaying function is given by

$$\lambda = \delta * \exp\left(\tfrac{-E}{|E|}\right) \tag{4}$$

where
δ is the initial learning rate
E is the current epoch count
$|E|$ is the total number of epochs

4.4.2 Support Vector Machine (SVM)

The PyML[2] library was used for SVMs. For the table detection problem, we used an SVM with a Gaussian Kernel of degree 5 and a C value of 20. For the table decompostion problem, we used an SVM with a Linear Kernel and a C value of 10. The hyperparameters were chosen by trial and error and even this seemed to give us a better classification accuracy. In future work, we plan to do a more formal search of the hyperparameters using Grid Search.

4.4.3 Logistic Regressor

We implemented a Logistic Regressor, modeling it as a single layer neural network with sigmoid output using Stochastic gradient descent. Decayed learning rate was used for training. The sigmoid was thresholded to 0.5 for prediction.

4.5 Table Detection

Liu et al.[4] observed that in most of the PDF documents, table lines followed a particular structure and are sparse. According to them, a line is sparse if the minimum space gap between pair of consecutive words is larger than a threshold (or) length of the line is much shorter than a threshold. We too observed the same behavior in the documents we examined and hence as proposed in [4], we use this sparse line property of table lines for table detection. In [4], the problem was posed as a sequence labeling problem where they labeled each line as either NONSPARSE or one of the different kind of SPARSE lines. We however, pose the problem as both a sequence labeling problem and a classic classification problem where each line is classified as either a candidate TABLE or NONTABLE line.

4.5.1 Post Processing

After the classifier identifies the candidate TABLELINEs, we postprocess it so that we can remove false positives and include those lines which are false negatives (i.e. lines which were TABLELINE but were classified as NONTABLELINE). We try to increase the recall of the table lines as much as possible as the NONTABLELINEs will however be removed (if irrelevant) in the table decomposition step. We adjust vari-

[2]pyml:http://pyml.sourceforge.net/

ous thresholds with this premise in mind. The postprocessing algorithm is as follows: The subroutine Find-

Algorithm 2 Postprocessing of table detection results

```
Input: Classification results for each line in the input document D
Output: A list of tables T in the input documents
for lines in document do
  if line begins with a keyword then
    data = FindPossibleTableStructureAfterThisLine(curindex)
    if data then
      Add table to T
  if line is TABLELINE then
    data = FindPossibleTableStructureAfterThisLine(curindex)
    if data is not null then
      keywordloc = IsTableKeywordAfterThisLine(curindex)
    if keywordloc then
      data = FindPossibleTableStructureBeforeThisLine(curindex)
      if data is not null then
        Add table to T
```

As of now, we only use table followed by a number (Eg: 'Table 1', 'Table 3.2') as our keyword. We plan to add more keywords like 'Figure' in future work.

-*FindPossibleTableStructureAfterThisLine Subroutine*

```
Data=[]
Sparseline = find next sparse line after current line
if difference between current line no and sparseline's line no ¿ threshold1 then
  return
Append the lines between current line and next sparseline to Data
while difference between current line no and next sparse line no ¡ threshold2 do
  Append lines to data
Return data
```

PossibleTableStructureAfterThisLine finds the closest sparse line to the current line and if this distance is less than a threshold, the whole block of lines between the line and the sparse line is returned. The subroutine FindPossibleTableStructureBeforeThisLine is similar to the FindPossibleTableStructureAfterThisLine subroutine except that the scan is carried bottom-up instead of top-down.

4.6 Table Decomposition

The table decomposition step takes the tables detected from the Table Detection component and identifies the Header and data rows in the table. We first attempt to classify rows in a table. We modeled this as a sequence labeling problem where a row in a table can be classified as 'HEADER 1", "HEADER 2", "DATA ROW", "DATA HEADER", "FOOTER" and "CAPTION". We use CRFs to classify the various table row components. We compare this approach with a linear SVM and a Logistic Regressor (LR) for classifying "HEADER" and "DATA" and evaluate their performance.

5 Experiments and Results

In this section, we demonstrate the experiments we ran for table detection and table decomposition and analyse the results.

5.1 Data Sets

One of the biggest challenges faced was that there was no off-the-shelf annotated data set available for this problem. So, we manually annotated our dataset. We took 15 PDF files taken at random from the publications page of CS faculty from the University of Wisconsin, Madison. Each PDF file was first converted to xml using pdftohtml. They were then passed through the preprocessing algorithm as described in Algorithm 1. Each xml preprocessed was converted to a HTML file which was hosted on a web server. The HTML file was designed in a way to allow the user to demarcate the table boundary. When the user clicked on the 'Submit' button on the page, a CGI script written in python read the HTML post data and wrote the annotated data to a file. Though, annotations for all these 15 pdfs were done by us, the main reason to create HTML files and publish them in a web server is two fold: (1)It is easy to annotate the training data on a web page for the format that is required by our system, (2) For annotating a large number of PDF documents (going forward), we can easily crowd source the effort. Crowd sourcing drastically reduces the time required to annotate data and is very cheap. There are, of course, many challenges associated with crowd sourcing and tackling them is beyond the scope of this paper.

The 15 PDF files contained a total of 16010 lines out

Table 1: Table Detection Using Conditional Random Fields - Confusion Matrix

	Non-Sparse	Sparse
Non-Sparse	14128	182
Sparse	201	1541
Learning Rate : 0.2 Number of Epochs : 80		

Table 2: Table Detection Using Support Vector Machines - Confusion Matrix

	Non-Sparse	Sparse
Non-Sparse	13376	934
Sparse	75	1667
Degree of Gaussian Kernel : 5 C = 20		

of which 14635 were NONTABLELINEs and 1375 lines were TABLELINEs. Also they contain 65 tables in total. For the table decomposition experiments, we annotated 50 tables where each row of the table is tagged as one of the following - 'NONTABLE TEXT' (incase there are false positives in the input), 'HEADER 1", "HEADER 2", "DATA ROW", "DATA HEADER", "FOOTER" and "CAPTION".

5.2 Table Detection

We trained models using 3 different Machine Learning techniques - CRF, SVM (Gaussian kernel, degree 5) and LR and evaluated their performance.

We adopt a 5 fold Cross Validation approach to empirically compare the 3 different learning settings - CRF, SVM and LR. The confusion matrix (indicating the actual vs predicted classes) for the 3 methods - CRF, SVM and LR are listed in Tables 1, 2 and 3 respectively. Also, we can infer the precision and recall for both the classes (TABLELINE and NONTABLELINE) from the Confusion matrices. The precision and recall for the NONTABLELINE class and TABLELINE class are listed in Table 4 and Table 5 respectively.

Table 3: Table Detection Using Logistic Regressor - Confusion Matrix

	Non-Sparse	Sparse
Non-Sparse	14120	190
Sparse	207	1535
Learning Rate : 0.2 Number of Epochs : 80		

Table 4: Precision and Recall of NONTABLELINE

Method	Precision (in %)	Recall (in %)
Conditional Random Fields	98.59	98.72
SVM Gaussian	99.44	93.41
Logistic Regression	98.55	98.67

Table 5: Precision and Recall of TABLE LINE

Method	Precision (in %)	Recall (in %)
Conditional Random Fields	89.43	88.46
SVM Gaussian	64.09	95.69
Logistic Regression	88.98	88.40

The reason for the relatively low precision and recall for the TABLELINE class are two fold: (1) Lack of a large number of TABLELINEs in our dataset. (2) Features engineered towards reducing the error of NONTABLELINEs instead of that of TABLELINEs. Nevertheless, if we add more training data with a lot of TABLELINEs and add in more features which are geared towards reducing the error rate of TABLELINEs, we are certain than the Precision and Recall of the TABLELINE class would improve.

5.2.1 Impact of PostProcessing

Even though the precision and recall of TABLELINEs is relatively low, when combined with our postprocessing algorithm (explained in Section 4.5.1) for Table boundary detection, surprisingly most of the tables and their boundaries are properly detected. This can be explained by the following reasons (1) A low precision and recall for TABLELINE class means that some of the TABLELINEs are missed and are classified as NONTABLELINEs. However, the precision and recall of the NONTABLELINEs is very high. This means that not many NONTABLELINEs are classified as TABLELINEs. (2) Since our postprocessing algorithm tries to include those lines classified as NONTABLELINEs in between TABLELINEs (which seems to be the major scenario in our case), the actual TABLELINEs which were misclassified as NONTABLELINEs are added back to the result. The precision and recall of

Table 6: Precision and Recall of TABLE LINE after Post-Processing

Method	Precision (in %)	Recall (in %)
Conditional Random Fields	89.75	95.52
SVM Gaussian	65.80	100
Logistic Regression	88.93	95.52

the Table Detection after post processing is presented in Table 6

Our initial assumption from the works of [4] was that sequence labeling using CRF should perform better than any classifier. But surprisingly, in our experiments we found that SVM with a Gaussian Kernel gave better detection of table boundaries than CRF and LR. Eventhough the precision of SVM is lower, the recall is very high relative to the other techniques which implies most of the TABLELINEs are retrieved after postprocessing leading to better table boundary detection. However, to conclusively say that SVM with a Gaussian Kernel is the best approach for this problem, we need to try out more experiments with carefully selected new features and varying the number of epochs and the learning rate of CRF and LR. We plan to do this as part of our future work.

5.3 Table Decomposition

For mutli-class classification of table row elements using CRFs, we adopt a 3 fold cross validations and found a precision of 77% for DATA ROW. But the precision for individual table header elements like 'HEADER 1', 'HEADER 2', 'DATA HEADER' was very low. We believe this is due to the lack of a larger training set and more diversity between tables in the current training set. We are currently working towards building a larger data set and test our results over them. We are also looking into the features that will help us improve the precision and recall.

We adopt a 6 fold Cross Validation approach to empirically compare the 2 different classifiers - linear SVM and Logistic Regression. The confusion matrix (indicating the actual vs predicted classes) for the 2 methods - SVM and LR are listed in Tables 7 and 8. The precision and recall for the DATA class and HEADER class are listed in Table 9 andTable 10 respectively.

Table 7: Table Decomposition Using SVM - Linear Kernel - Confusion Matrix

	DATA	HEADER
DATA	414	31
HEADER	26	44
C=10		

Table 8: Table Decomposition Using Logistic Regressor - Confusion Matrix

	DATA	HEADER
DATA	405	40
HEADER	20	50
Learning Rate : 0.2 Number of Epochs : 50		

Table 9: Precision and Recall of DATA

Method	Precision (in %)	Recall (in %)
SVM Linear	94.09	93.03
Logistic Regression	95.29	91.01

Table 10: Precision and Recall of HEADER

Method	Precision (in %)	Recall (in %)
SVM Linear	58.67	62.86
Logistic Regression	55.56	71.43

We infer that both the binary classification methods perform almost similarly. However, we have only tried on a linear SVM with C = 10. We need to perform Grid search to find out the kernel and the hyperparameters that give us the best accuracy given the feature set, which we consider in our future work. Also, the reason for the very low precision and recall for the HEADER class are two fold: (1) Lack of a large number of HEADERs in our dataset. (2) Features engineered towards reducing the error of DATAs instead of that of HEADERs. Nevertheless, if we add more training data with a lot of HEADERs and add in more features which are geared towards reducing the error rate of HEADERs, we are certain than the Precision and Recall of the HEADER class would improve. We also want to try a multi-class SVM to predict intricate table row elements such as 'HEADER 1', 'DATA HEADER', 'DATA', 'CAPTION' etc.

6 Conclusion

In this project, we have built a prototype of a table extraction system. We empirically tested the table detection and table row decomposition on a sequence labeler and two classifiers and have shared our experimental results. We learnt from this problem that a purely heuristic solution or a purely Machine Learning technique based solution perform poorly when compared with a hybrid approach involving both heuristics and Machine Learning (ML) techniques. Also, we learnt that feature selection and evaluation plays a vital role in the performance of a ML technique. Going forward, we plan to add in more features and evaluate the performance. We also plan to use Schema Matching techniques to identify the significance of the detected rows/columns so that the information in the data rows become more relevant in the context of an Information Extraction (IE) system.

7 Open Source Code

The implementation is open source and is located in GitHub https://github.com/shriram-sridharan/TableExtraction. Please read the README.txt file for the dependencies and the procedure to execute the code. We greatly encourage and appreciate any feedback on the project.

References

[1] J. Fang, P. Mitra, Z. Tang, and C.L. Giles. Table header detection and classification. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[2] J. Lafferty, A. McCallum, and F.C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[3] Y. Liu, K. Bai, P. Mitra, and C.L. Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 91–100. ACM, 2007.

[4] Y. Liu, P. Mitra, and C.L. Giles. Identifying table boundaries in digital documents via sparse line detection. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1311–1320. ACM, 2008.

[5] D. Pinto, A. McCallum, X. Wei, and W.B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242. ACM, 2003.

[6] Burcu Yildiz, Katharina Kaiser, and Silvia Miksch. pdf2table: A method to extract table information from pdf files.