Abstract—Tables are very commonly used to present relational data. This report focuses on mining structured data from markup language specified tables. Table recognition, table interpretation and presentation of results are discussed. First, two categories of features are developed to recognize genuine tables. These recognized tables provide knowledge of table types we need in order to synthesize tables for training. After synthesizing tables with random structures and labelling in a hierarchical way, we experimented with several models on predicting labels for unseen tables. Then we explored potential applications of the extracted table.

Keywords—Information Extraction, Web Mining, Character-level prediction.

I. INTRODUCTION

Tables, which are simple and easy to interpret, are very commonly used to describe schedules, organize statistical data, and summarize experimental results, in texts of many different domains. Because tables provide both unstructured content as well as structured semantic information about how content from different cells relate, extracting both table content and cellular structure an facilitate many downstream applications such as document understanding, question-and-answering, text retrieval, etc.

Meanwhile, markup languages like HTML provide very flexible constructs for specifying tables. For example, figure 1 shows an example table described with HTML. In general, an HTML table begins with an optional caption followed one or more rows. Each row is formed by one or more cells, which are classified into header and data cells. Cells can be merged across rows and columns. The following tags are used:

1. <table ...> </table>
2. <tr ...> </tr>
3. <td ...> </td>
4. <th ...> </th>
5. <caption ...> </caption>

They denote main wrapper, table row, table data, table header, and caption for a table. Another table specified in XML is shown in Figure 2. In this case, specific tags and conventions are not required to specify a table. Users can use any tag they want and maintain a table structure by nesting open and closing tags. But the flexibility also means that extracting knowledge from tables specified with any markup language is harder than extracting data from one single markup language with tables specified using one single format. Typical attempts to scrape tables from the web will use regular expressions or parsers derived from context-free grammars in order to parse tables such as the one in Figure 1.

But in practice, tables are not restricted to the tags in Figure 1. The set of tags, row separators, column separators, and other constructs used to specify the units needed to compose tables is infinite. For example, even in a single markup language such as HTML, instead of a <tr> tag, a web developer could use something like <div style="table_row">. Clearly, any attempt to programmatically parse tables by recognizing specific tags or separators will necessarily suffer from recall issues, since the space of such tags and separators is unbounded. Thus, we seek to answer the question: can we parse the content and cell structure of tables in a manner that is independent of the exact tags or separators used, instead relying on structural commonalities in how all tables are specified in markup?

Hurst [10] is the first attempt to collect a corpus from HTML files, \LaTeX files and a small number of ASCII files for table extraction. This paper focuses on extracting tables from any markup languages by using a language independent model.

The remaining of this report is organized as follows. Section II-C introduces the table recognition problem, where we discussed features and algorithms we used to distinguish genuine tables from random markup noise. Section III gives the definition of the table interpretation problem and explored the models we evaluated. Section IV discusses our evaluation and empirical results.

Finally, we briefly surveyed related work in Section VI and conclude this study in Section VIII.

II. TABLE RECOGNITION

After the stages of hypertext processing and table filtering, the remaining candidates are sent to table recognition module for further analyses. The major techniques in this module can be classified into three groups based on the algorithms, including (1) Label-based, (2) Context-based and (3) Feature-based algorithms. Our research chooses feature-based algorithm due to its accuracy and simplicity.

A. Label-based

The HTML format is a hierarchical markup language where the markup tags define the layout of the information. The \(<TABLE>\) tag is used in HTML documents to construct tables. However, in spite of tags demarcating a cell boundary in HTML tables, it is still difficult to automatically identify the labels for each data cell, as the tables are designed for visual interpretation and not for automatic extraction. One of the important contributions of paper [14] is learning labels and using them for structure recognition.

1. Learning Labels

(1) Learning Labels
Tables have two types of information: the label information and the data. The labels are the attributes of the data. If the some of the labels of these tables are known, then this information can be used to identify the structure of the tables, which in turn will help assign the right labels to the data cells. Since tables are a semi-structured form of representing data, the labels are present in a regular format where consecutive label cells appear either in a row or a column. Locating the known labels in a table will help differentiating the label rows and columns from the data rows and columns.

(2) Heuristics for Table Extraction

The algorithm starts by parsing the content in the `<TABLE>` tag. It uses a set of heuristics in conjunction with the labels learnt from the examples, to recognize the structure. The heuristics used to recognize the structure are given below:

- **Span tag**: The `<td>` tags in HTML can contain the attribute `span` which specifies the number of row cells or column cells that the current cell spans. We recognize the span attribute in the tags and use this information to assign the spanning cell to multiple rows and columns.

- **Rows with empty data cells**: After separating the label cells from the data cells, we identify rows with empty data cells. If the previous data row is not empty and the next data row is also not empty, then the label of the current row is a super row label. A super-row label is one which spans over the row labels below it.

- **Single row cell in a row**: If there is only one row cell in a row and that cell contains a label, then naturally the label is a super row label.

- **Single empty data cell**: If there is only one data cell in a row and the cell is empty then the label for that row is a super row label.

B. Context-based

The context among cells in table is important. This technique is well studied in the paper [4]. We agreed with that value cells under same attribute names demonstrate similar concepts. The following metrics are employed to measure the cell similarity.

(1) **String similarity**

We measure how many characters are common in neighboring cells. If the number is above the threshold, the two cells are similar.

(2) **Named entity similarity**

The metric considers semantics of cells. A rule-based method similar to the paper [3] is employed to tell if a cell a specific named entity. The neighboring cells belonging to the same named entity category are similar.

(3) **Number category similarity**

Number characters (0-9) appear very often. If total number characters in a cell exceeds a threshold, we call the cell belongs to the number category. The neighboring cells in number category are similar.
We count how many neighboring cells are similar. If the percentage is above a threshold, the table tags are interpreted as a table.

C. Feature-based

In our case, we need to find a combination of features that together provide significant separation between genuine and non-genuine tables while at the same time constrain the total number of features to avoid the curse of dimensionality. Past research has clearly indicated that layout and content are two important aspects in table understanding [11]. Those features were designed to capture both of these aspects. In particular, we leverage 16 features from [16] which can be categorical into three groups: seven layout features, eight content type features, and one word group feature. We attempt to capture the global composition of tables as well as the consistency within the whole table and across rows and columns.

1) Layout Features

In HTML documents, although tags like <TR> and <TD> may not be reliable indicators of the number of rows and columns in a table. Variations can be caused by spanning cells created using <ROWSSPAN> and <COLSPAN> tags. Other tags such as <BR> could be used to move content into the next row. Therefore, to extract layout features reliably one can not simply count the number of <TR>’s and <TD>’s.

Given a table T, assuming its number of rows and columns are rn and cn respectively, we compute the following layout features:

- Average number of columns, computed as the average number of cells per row:
  
  \[ c = \frac{1}{rn} \sum_{i=1}^{rn} c_i, \]

  where \( c_i \) is the number of cells in row \( i \), \( i = 1, \ldots, rn \);

- Standard deviation of number of columns:

  \[ dC = \sqrt{\frac{1}{rn} \sum_{i=1}^{rn} (c_i - c) \cdot (c_i - c)}; \]

- Average number of rows, computed as the average number of cells per column:

  \[ r = \frac{1}{cn} \sum_{i=1}^{cn} r_i, \]

  where \( r_i \) is the number of cells in column \( i \), \( i = 1, \ldots, cn \);

- Standard deviation of number of rows:

  \[ dR = \sqrt{\frac{1}{cn} \sum_{i=1}^{cn} (r_i - r) \cdot (r_i - r)}. \]

- Average overall cell length:

  \[ cl = \frac{1}{en} \sum_{i=1}^{en} cl_i, \]

  where \( en \) is the total number of cells in a given table and \( cl_i \) is the length of cell \( i \), \( i = 1, \ldots, en \);

- Standard deviation of cell length:

  \[ dCL = \frac{1}{en} \sum_{i=1}^{en} (cl_i - cl) \cdot (cl_i - cl); \]

- Average Cumulative length consistency, CLC. In particular, this feature is designed to measure the cell length consistency along either row or column directions. It is inspired by the fact that most genuine tables demonstrate certain consistency either along the row or the column direction, but usually not both, while non-genuine tables often show no consistency in either direction.

2) Content Type Features

Web documents are inherently multi-media and has more types of content than any traditional documents. For example, the content within a <Table> element could include hyper-links, images, forms, alphabetical or numerical strings, etc. Because of the relational information it needs to convey, a genuine table is more likely to contain alpha or numerical strings than, say, images. The content type feature was designed to reflect such characteristics and including: hyperlink, alphabetical, digit, empty, others and average content type consistency (CTC). This last feature is similar to the cell length consistency feature.

3) Word Group Feature

If we treat each table as a "mini-document" by itself, table classification can be viewed as a document categorization problem with two broad categories: genuine tables and non-genuine tables. We designed the word group feature to incorporate word content for table classification based on techniques developed in information retrieval [12][18].

After morphing [13] and removing the infrequent words, we obtain the set of words found in the training data, W. We then construct weight vectors representing genuine and non-genuine tables and compare that against the frequency vector from each new incoming table.

Given a new incoming table, let us note the set included all the words in it as \( W_n \). Since W is constructed using thousands of tables, the words that are present in both W and \( W_n \) are only a small subset of W. Based on the vector space model, we define the similarity between weight vectors representing genuine and non-genuine tables and the frequency vector representing the incoming table as the corresponding dot products.

III. TABLE INTERPRETATION

A. Problem Definition

We now describe the problem we aim to solve.

Problem 1: (MARKUP-AGNOSTIC TABLE INTERPRETA-

tion) Given a table \( T \) from an arbitrary markup language \( \mathcal{L} \), extract the content of the table at the granularity of individual cells into a unified format such as the following:
An alternative method to hidden Markov model is **Structured Perceptron**, which is a discriminative training methods use perceptron machines for Hidden Markov Models[5].

**Problem 2: (CHARACTER-LEVEL LABEL PREDICTION)**

Given a training corpus $C$, which contains training instances $\{C_1, C_2, ..., C_n\}$. Each training instance can be treated as token span $t_1, t_2, ..., t_n$ and we assume the separator token has been give, e.g. open tag `<` and close tag `>`. The problem is formalized as, for each training instance, we have label sequence $y = \{y_1, y_2, ..., y_i\}$, where $y_i$ indicates hierarchy of corresponding character as we depicted in data generation section. Our task is utilizing effective sequence prediction model to predict output label sequence precisely.

**B. Models**

1) **Traditional Sequence Prediction**: Most machine learning algorithm are designed for independent, identically distributed data. However, in this case, we are looking for algorithm that can handle long term dependencies and feature free model.

A hidden Markov Model has been extensively studied by assuming Markov process with hidden states. It is a subclass of Bayesian networks known as dynamic Bayesian Networks. In HMM, we assume current token can influence future labeling in the future. HMM generate the observed sequence by both input and hidden state. We concatenate training instances $X$ as $X = x_1, x_2, ..., x_i$, each subsequence $x_i$ is one training table instance we defined above. The observed sequence $Y$ is tokenwise label we generate, we would like to point out that the output sequence itself contains hierarchy information, which help us reconstruct the data. We define transition probability as follows,

$$P(Y) = \sum P_t(Y_i|Y_{i-1})$$

$$P(X|Y) = \sum P_t(X_i|Y)$$

where $P_t$ is the transition probability between different labels and $P_t$ is the generation probability given current label $Y_i$, an illustrative figure is given in Fig. 3, $Y_i$ is the observed sequence and $X_i$ is input sequence.

An alternative method to hidden Markov model is **Structured Perceptron**, which is a discriminative training methods use perceptron machines for Hidden Markov Models[5].

2) **Neural Network based sequence prediction**: Neural Network based models are widely used in time sequence prediction recently. We will carefully compare and analyse several promising variants of Neural Network methods.

**Character Embedding** Since our model don not expect any feature engineering work, whereas neural network based models usually require fix number of input neurons. We consider one hot embedding and embedding neuron layer for experiments. In our setting, the dimension of dictionary is pretty small 100 dimension.

For example, one hot embedding, if we have character dictionary $\{a, b, c, d\}$, coming sequence $a, b, c, d$ can be converted to five feature vectors with one hot at corresponding location. For example, the one hot embedding for character a is 1, 0, 0, 0

The other popular embedding method word2vec[8] is using embedding layer and backpropagate whole network to learn this embedding layer. We note that, our model takes character sequence as input, thus the embedding layer learn character semantic meaning rather than word. We investigate embedding layer illustrated in Fig.4 has better performance than straightforward one hot embedding.

**Multi-layer Perceptron** Neural Network is not designed specifically for time serious prediction Work in neural networks has concentrated on forecasting future developments of the time series from values of $x$ up to the current time. Specifically, we need to manually select time window, and parameters complexity is therefore, depends on the context window $d$. Formally, we would like to learn function $F$, a mapping from input vector space to output space.

For this multi-layer perceptron model, see Fig 5, we design a model with two hidden layer with 64 dimensions, we also apply dropout techniques, which is proved to be effective to avoid overfitting.

$$y(t) = F(x(t), x(t-1), ..., x(t-d))$$

**Many-to-Many Prediction Model** We notice that traditional multi-layer perceptrons can not fit in time series prediction well. And researchers developed family of recurrent neural networks, which is designed for sequence prediction. Specifically, for these methods, we model our problem as "many-to-many" prediction, see Fig.6 which means for each training instance, we will have multiple labels.

**Recurrent Neural Network** Recurrent Neural Network creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike traditional feed forward networks. RNNs can use their internal memory to process arbitrary sequences of inputs. Fig.7 shows a basic
Fig. 5. Example of multi-layer perceptrons with two hidden layers.

Fig. 6. Many to Many Schema architecture of recurrent neural network. The way RNN carry memory forward is as follows:

\[ h_t = \Phi(Wx_t + Uh_{t-1}) \]  

(3)

The hidden state at time step \( t \) is \( h_t \). It is a function of the input at the same time step \( x_t \), modifies by weight matrix \( W \) and hidden-state-to-hidden-state matrix \( U \). This combination is also the main difference between RNN and feed forward networks.

**Long Short Term Memory** LSTMs[9] contain information outside the normal flow of the recurrent network in a gated cell. The cell can forget its state, or not; be written to, or not; and be read from, or not, at each time step. Different sets of weights filter the input for input, output and forgetting. The forget gate is represented as a linear identity function, because if the gate is open, the current state of the memory cell is simply multiplied by one, to propagate forward one more time step. With this forget state, the LSTMs model is more flexible compared with RNN because it can forget some useless information. Therefore, we believe LSTMs better captures long distance dependencies. And it do show superior performance in our experiment section.

IV. EXPERIMENTAL RESULTS

We conduct various experiments to study effectiveness of the potential methods described before. Also we generate large dataset of synthetic web pages and proposed one annotation method that can easily translated into hierarchy structures and help model learning long distance dependencies.

**A. Data Preparation**

As it is not realistic to manually label a large number of html or other mark-up language specified texts, we decide to synthesize html style data with labelling. We first generate simplest table with traditional tags without any advanced attributes. Figure 9 and Figure 11 shows the corresponding html text and labelling. Then we go on to generate more complex tables with more attributes in tags. Figure 10 shows an example of this. Specifically, we used Gaussian distribution to generate the number of rows and columns. For the content of each cell, we use sentences composed of random words so far. For the attributes, we first extract a set of attribute value pairs, for example, attribute “border” and value “0” are extracted from one table. Then we randomly sample attributes and their values from this set to compose more complex tables.

**B. Experimental Results**

In our experiments, we compare following methods that we discussed before:

1) **HMM**: hidden Markov Model
2) **Structured Perceptron**
3) **MLP one hot**: Multi-Layer Perceptron using one hot layer, with two hidden layers and 64 dimensions, we apply dropout techniques and set dropout ratio as 0.5
4) **MLP embedding**: Multi-Layer Perceptron using embedding layer
5) **RNN**: Vanilla Recurrent Neural Network with 32 hidden units
6) **GRU**: Gated Recurrent Network. The GRU unit controls the flow of information like the LSTM unit, but
### Methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>74.02%</td>
</tr>
<tr>
<td>Structure Perceptron</td>
<td>74.02%</td>
</tr>
<tr>
<td>MLP one hot</td>
<td>74.02%</td>
</tr>
<tr>
<td>MLP embedding</td>
<td>74.01%</td>
</tr>
<tr>
<td>RNN</td>
<td>82.23%</td>
</tr>
<tr>
<td>GRU</td>
<td>98.37%</td>
</tr>
<tr>
<td>LSTM</td>
<td>99.39%</td>
</tr>
</tbody>
</table>

| TABLE I. | EVALUATION RESULT FOR MULTIPLE METHODS |

In experiments, we conduct experiments on our 5000 synthetic web page datasets and we set max hierarchy depth as 10. Thus, in our softmax output layer, there will be ten output unit to classify inputs into 10 categories. We use 80%/20% train-test splition and we report categorical classification accuracy on test set in Table IV-B. From result, we can find traditional Markov based methods only predicts 30% of test label correctly. It’s reasonable because we are doing character level sequence prediction, the open tag and close tag can be separated by hundreds of characters.

Among neural network based methods, LSTM and GRU have better accuracy since they encode more context information, especially LSTM holds 99% accuracy on test set. We believe the forget gate mechanism makes LSTM capable of learning long distance coherence during training stage. Vanilla RNN can not fit training data well, but it’s still much better than MLP, since RNN is a recurrent structure yet MLP only hard encode time distribution. We also plot first 50 epochs in neural network training in Fig.12, we can also find RNN and MLP fails to converge to a small loss.

#### V. POTENTIAL APPLICATIONS

In this section, we will analyze a number of applications that will potentially benefit from the extracted table structures. The prediction for table in 1 is as following:

0 0 Month 0 1 Savings
1 0 January 1 1 $100
2 0 February 2 1 $80
TABLE II. ATTRIBUTE VALUE PAIRS FROM EXTRACTED LABELS

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(January, Savings)</td>
<td>$100</td>
</tr>
<tr>
<td>(February, Savings)</td>
<td>$80</td>
</tr>
</tbody>
</table>

With the location labels, (0,0) (0,1), (1,0), (1,1), (2,0) and (2,1) in this example, we can locate the data in column wise or in row wise depending on the interpretation. We can concatenate two attributes together, for example, $100 at (1,1) can be located to (0,1) by reading in column wise, which denotes Month and (1,0): January by reading row wise. Then $100 can be perceived as the value of attribute January and Savings. The extracted attribute value pairs are summarized in table V. This can be further utilized in an application like Question and Answering system. For example, given a query like how much is the savings of January, after consulting the extracted data, we can get the value $100.

VI. RELATED WORK

The subfield of information extraction has been around almost as long as the web itself, and prior work has studied table recognition and extraction from various angles. The works closest in spirit to this paper [6], [7] involve a task known as wrapper induction, whereby extraction frameworks attempt to learn a set of rules which end up specifying a grammar for parsing tables and fields out of tables. This fundamentally differs from our approach in that the extracted information is typically specific to a single field or set of fields, whereas we are concerned with parsing out the overall cell structure.

Other works [17], [15] are specific to HTML, attempting to differentiate when \(<table>\) tags are used to describe tables, and which tables are (mis)used to specify page layout, as well as to determine which tags refer to headers and which headers apply to different content cells. These differ from our approach in that they are restricted to one markup language (HTML).

WebTables [2] attempts to determine which tables on the web contain high-quality relational data, and then automatically build schemata for said tables. Our methods could be used, for example, as a preprocessing step for WebTables, providing high-quality structured data which can then be used for schema induction.

In [1], the authors are not concerned with extracting markup-specified tables, but with identifying pages that follow a common template, from which a relational schema can then be constructed and tables populated. While an interesting problem in its own right, the extraction task in that paper operates on a subset of data completely disjoint from the one we are interested in.

VII. FUTURE WORK

There are still other spaces to improve performance. The cues from context of tables and the traversal paths of HTML pages may be also useful. In the text surrounding tables, writers usually explain the meaning of tables. For example, which row (or column) denotes what kind of meanings. From the description, we can know which cell may be an attribute, and along the same row or column we can find their value cells. Besides that, the text can also show the semantics of the cells. For example, the table cell may be a monetary expression that denotes the price of a tour package. In this way, even money marker is not present in the table cell, we can still know it is a monetary expression.

Note that HTML texts can be chained through hyperlinks like previous and next. The context can be expanded further. Their effects on table mining will be studied in the future. Besides the possible extensions, another research line that can be considered is to set up a corpus for evaluation of attribute-value relationship. Because the role of a cell (attribute or value) is relative to other cells, to develop answering keys is indispensable for table interpretation.

Finally, we would like to explore more relational data derived from more types of mark up languages. Potential data sources that researchers have studied include tabular layouts that do not use the table tag, deep web databases, socially-tagged data items, HTML-embedded lists, and natural language text.

VIII. CONCLUSION

In this report, we proposed a systematic way to mine tables from markup language specified text. Table recognition, table interpretation and potential application of the labelling result are discussed. Specifically, an intuitive but effective way of labelling table cells are developed. And the performance of several models in predicting labels are compared. Our model based on Long short term memory actually performs all other models and shows a prediction accuracy of 99.39%.

REFERENCES


