Schema Extraction for Tabular Data on the Web *

Marco D. Adelfio Hanan Samet

Center for Automation Research, Institute for Advanced Computer Studies
Department of Computer Science, University of Maryland
College Park, MD 20742 USA
{marco, hjs}@cs.umd.edu

ABSTRACT

Tabular data is an abundant source of information on the Web, but remains mostly isolated from the latter's interconnections since tables lack links and computer-accessible descriptions of their structure. In other words, the schemas of these tables — attribute names, values, data types, etc. — are not explicitly stored as table metadata. Consequently, the structure that these tables contain is not accessible to the crawlers that power search engines and thus not accessible to user search queries. We address this lack of structure with a new method for leveraging the principles of table construction in order to extract table schemas. Discovering the schema by which a table is constructed is achieved by harnessing the similarities and differences of nearby table rows through the use of a novel set of features and a feature processing scheme. The schemas of these data tables are determined using a classification technique based on conditional random fields in combination with a novel feature encoding method called logarithmic binning, which is specifically designed for the data table extraction task. Our method provides considerable improvement over the well-known WebTables schema extraction method. In contrast with previous work that focuses on extracting individual relations, our method excels at correctly interpreting full tables, thereby being capable of handling general tables such as those found in spreadsheets, instead of being restricted to HTML tables as is the case with the WebTables method. We also extract additional schema characteristics, such as row groupings, which are important for supporting information retrieval tasks on tabular data.

1. INTRODUCTION

For structured data that is never stored in a database system, a document containing a *data table* (e.g., a spreadsheet) may be the closest it comes to being in a machine-readable format. Consisting of textual values in a two-dimensional grid format, data tables owe their appeal primarily to their high information density. In other words, due to the semantic meaning communicated in their layout and structure, the need for descriptive words is minimized, allowing tables to communicate more information than prose in the same amount of space. The power of data tables is evident in their abundance in published documents of all types on the Web, including spreadsheets, HTML tables in web pages, tables embedded in PDF documents, and many others. In essence, these files can collectively be viewed as an enormous distributed database (an informal web search at the time of this writing returns over 50 million spreadsheet results and prior work has found over 14.1 billion HTML tables [2]). However, the implicit or visual structures employed in data tables are not easily machine-recognizable, and consequently this database lacks most of the features that enable the efficient exploration and querying abilities of a standard DBMS.

Numerous research efforts have attempted to extract structure and information from data tables, such as the well-known technique of Google's WebTables, and we examine those efforts in more detail in Section 2. In general, these approaches rely on the high quantity of data tables that have simple structure, in order to avoid dealing with those data tables with more complex structure. We say a table has *simple structure* if it consists of one row of header values, followed by one or more rows containing data values. The values in the header row describe the domain of values in the data rows beneath it, while the data rows contain tuples that roughly correspond to rows in a relational database. Many tables exhibit simple structure, which may justify the focus placed on them by WebTables and others. However, many tables are more complex. As one example of a more complex structure, tables that contain geographic data about cities are commonly organized into groups by state/province. These groups may be designated in several ways, but one popular technique is to insert a *group header row* containing the name of the state/province as a single value, directly above the data rows that it includes. This row is not a data row like the others in the table. Our approach involves classifying each row based on its function to guide our extraction process. The row functions that we consider are given in Section 3. In addition to achieving high accuracy rates when classifying the function of individual rows, our focus on handling complex structure results in very high accuracy rates for correctly interpreting full tables.

To guide the structured data extraction process, it is helpful to understand why explicit structural information is not present in these documents. The table extraction task is challenging because explicit structural information is not present in the table formats that hold them, due to an essential difference between data intended for computers and

*This work was supported in part by the NSF under Grants IIS-08-12377, CCF-08-30618, IIS-09-48548, IIS-10-18475, and IIS-12-19023, and by Google Research.

Proceedings of the VLDB Endowment, Vol. 6, No. 6

that intended for humans. While tabular data intended for computers (e.g., XML, RDF) is published in formats that communicate the structure explicitly, in well-specified ways that allow the data's schema to be easily accessed by algorithms [14], data intended for humans (e.g., spreadsheets, PDF, DOC files) contains structural information that is typically communicated visually (i.e., implicitly), such as the positioning, styling, and content of titles, column headers, and data. Consequently, there is no straightforward method for directly retrieving the structured data that human-oriented documents store. Although many data publishers recognize this and provide their data in multiple formats to allow for computer or human consumption, a large number of datasets still exist only in human-oriented formats, and thus lack the necessary metadata for querying.

In order to address these shortcomings, we developed a fully automated method for extracting implicit metadata from tabular data on the Web. Our technique utilizes visual attributes of table cells as input for a trained classifier based on conditional random fields [13], a powerful framework for sequence labeling tasks. The classification procedure involves extracting attributes of individual cells, combining them using a novel logarithmic binning method that we introduce, and processing them with the classifier. The classifier's output allows us to discern between relational and non-relational tables, and furthermore to identify *schemas* for the relational ones — that is, to classify the constituent rows of each relational table as either header rows, data rows, non-relational metadata rows, or rows of a few other types that we have identified. As our experimental evaluation reveals, we improve on the commonly-used WebTables extraction method by accounting for more complex table structure so that any row can be recognized as a header row. We perform tests on a new corpus of spreadsheet tables and HTML tables that we crawled from the Web, along with an existing corpus collected by other researchers [16]. In tests on both our new corpus and the existing corpus, the ability of our method to correctly interpret full tables without error is particularly apparent when compared to alternatives.

Our work includes the following contributions. First, we define a set of row classes that encompasses a broader range of row functions than previous work, enabling more complete post-processing techniques for using the data in other applications. Second, we apply a novel logarithmic binning scheme to encode collections of individual cell attributes as row features and show that this performs better than techniques presented in previous work [2, 19]. Third, in addition to HTML tables, we apply table extraction techniques to spreadsheets, which we believe are a more challenging, but potentially more rewarding, source of tabular data. Finally, our evaluation shows substantial improvements over alternative methods, especially when measuring our method's ability to correctly interpret full tables (that is, correctly classifying every row in a table with no errors). We believe that these aspects of our approach, in combination, provide the best available technique for extracting structured data from the enormous collection of available data tables.

The rest of this paper is organized as follows. Section 2 surveys prior work and describes our chosen classification method, conditional random fields. Section 3 presents our schema extraction method while Section 4 discusses common table patterns and their relevance for table extraction accuracy. Section 5 provides an evaluation of our method's application to spreadsheets and HTML tables. Section 6 highlights the benefits of our approach and adds concluding remarks.

2. RELATED WORK

2.1 Table Extraction

Many techniques for processing data tables have been presented (e.g., [5, 9, 12, 15, 17, 19, 23, 24]. See [8] for a survey). The initial challenge of processing tables is to determine whether a given table contains usable data that is in a suitable format for extraction. In database terms, we call such tables *relational* and other tables *non-relational*. Relational tables are defined by the relational model introduced by Codd [6], wherein a relational table consists of a set of attributes (typically found in a header row, such as "Name, Gender, Age") and a collection of one or more "tuples" (represented as data rows where each cell value is a member of its column's attribute domain, such as "John, Male, 30"). Tables containing relational data can also include values that do not fit into the relational model (i.e., non-relational values), such as a title, blank rows, aggregations (e.g., subtotals), and more. In this paper, we regard these tables as relational, whereas most prior work has treated these tables as non-relational. In our view, non-relational values that serve to supplement a relational table should not disqualify the table from being relational. As an example, we do not want to discard tables in which the first row contains a table title as non-relational. However, there are also many tabular grids that are created for purposes other than displaying relational data, such as HTML tables used for page layout and spreadsheets used as forms. These table types do not contain relational data that can be processed using relational operators such as projection or selection, so they should be discarded.

Several techniques focus on the relational/non-relational decision problem. Chen, Tsai, and Tsai [5] count the number of cells that are similar to their neighbors in terms of text length and datatypes, and use thresholds on these values to perform relational/non-relational classification on a set of tables from airline web sites. Instead of using a rule-based approach, Wang and Hu [24] treat the task of identifying relational data tables as a machine learning classification problem. They develop several features based on the layout of an HTML table and the distributions of cell lengths and content types, which serve as inputs to decision tree and support vector machine classifiers. In other work, relational tables are alternatively called "genuine tables" [24] or tables containing "true relations" [3].

Other methods go further and attempt to categorize the data found within data tables based on its function. The well-known approach of WebTables [2, 3, 4] builds upon prior work by using a machine learning classifier to distinguish between relational and non-relational HTML tables on the Web. The classifier is tuned for recall over precision, counting on subsequent processing to filter out non-relational table values. For relational tables, it attempts to extract any embedded metadata such as the labels and types of the columns, thereby learning the schema for the table. However, in WebTables, only the first row of a table is considered a candidate for the table header. In essence, this is due to the simple table structure assumed by WebTables and derived methods, including the work of Limaye et al. [16], which extracts relations from HTML tables, but depends on HTML formatting methods to indicate header cells. As we show in our evaluation, such an assumption discards a significant number of tables that have more complex structures. Our approach is designed to handle such complexities, thereby increasing the number of tables available for systems that process them. Some work on table extraction also relies on external knowledge sources, such

as YAGO [16] or custom fact databases extracted from text on the Web [23]. Our approach does not include such data, thereby allowing it to run accurately even on tables with contents that are not so globally relevant that they appear in knowledge bases.

Systems that rely on techniques like these are proliferating. Such systems use data extracted from data tables for different purposes, from search engines over structured data documents [18], to database augmentation using "facts" garnered from web tables [25], to automated ontology construction [12]. These systems handle tables with simple structure only, and consequently exclude large quantities of valuable table data. The approach presented in this paper is more comprehensive, and would provide larger and cleaner sets of data tables for applications such as these, or more general attempts at schema matching across a heterogeneous collection of data tables from the web, such as those introduced by Bernstein et al. [1].

2.2 Conditional Random Fields

Conditional random fields (CRFs) [13] are undirected graphical models introduced by Lafferty et al. that can serve as classifiers for sequence labeling tasks. Frequently used for natural language processing, such as part-of-speech tagging, CRFs have become a popular technique showing accuracy improvements over Hidden Markov Models (HMMs) and Maximum Entropy Models (MEMs) in many scenarios [13, 19].

CRFs are designed to maximize the probability of a sequence of labels Y, given an observation sequence X. The estimated joint probability $P(\boldsymbol{Y}|\boldsymbol{X})$ is defined to be

$$\frac{1}{Z(\boldsymbol{X})} exp\left(\sum_j \lambda_j f_j(\boldsymbol{Y}_{i-1}, \boldsymbol{Y}_i, \boldsymbol{X}, i) + \sum_k \mu_k g_k(\boldsymbol{Y}_i, \boldsymbol{X}, i)\right).$$

Here i is an index for the sequences X and Y. $F = \bigcup_j f_j$ and $G = \bigcup_k g_k$ are families of binary feature functions. Although arbitrary functions may be used, we employ the independence assumptions of *linear chain* CRFs, where each f_j is active for a distinct bigram of labels (i.e., a distinct pair of consecutive labels) and each g_k is active for a specific combination of a label and observation. Values λ_j and μ_k are estimated model parameters that are computed from training data. Normalization factor $Z(X)$ ensures that the probabilities of all label sequences sum to 1 for a given observation sequence X. As Lafferty et al. showed, computing $Z(X)$ is tractable without considering every possible observation sequence [13].

In the data table scenario, X represents the list of rows in the table, and Y represents the corresponding row classes. As an example, a transition function f_j may correspond to the transition from a header row to a data row, while a state function g_k could be active for header rows that contain no numeric values (the row features used by our method are described in Section 3.3).

Training a CRF model involves estimating the λ_j and μ_k parameters using one of several training methods. In this paper, we use the limited-memory BFGS algorithm [22], an iterative procedure that scales well for large datasets.

After a CRF is trained, classifying a new sequence is straightforward and fast, using a dynamic programming approach. In previous work, Pinto et al. [19] applied CRFs to the task of labeling rows in data tables, but limited their focus to ASCII tables from plain text documents. ASCII tables are useful, but we believe the quantity of other sources of data tables and the ease of identifying and extracting data from HTML tables and spreadsheets makes them more intriguing targets. Their study includes many row types that are specific to ASCII documents, while other generic row types, such as aggregate rows (e.g., containing a subtotal) are not treated separately from data rows. In addition, the variety of structures that are possible in other formats introduces different extraction challenges that we discuss later in this paper. In Section 3, we detail our method for using CRFs in the table extraction task.

3. SCHEMA EXTRACTION METHOD

The crux of our technique involves extracting relevant *row features* to represent each row of a data table, and then classifying each row using one of several *row classes* that we have defined. As with natural language sentences, data tables are constructed in accordance with general principles that make it easy for viewers to interpret the data without explicit instructions for how to do so. These widely-understood principles allow the communication of schema information based on implicit rules for formatting and positioning of data cells. We introduce a collection of features that preserves the formatting information present in data tables and a collection of row classes that more formally encapsulates these principles. A system built on these components can extract schema information that describes the structure of a data table in a fully automated manner.

To find data blocks and extract their schemas, we employ techniques similar to, though more general than, those used by WebTables [3]. WebTables determines whether a given table is relational by using machine learning techniques to train a classifier that labels tables as relational or non-relational, based on several holistic features of the table. In WebTables, tables classified as non-relational are discarded. Then, tables classified as relational are subjected to additional machine learning-based classification to detect whether the first row of the table serves as a header for the table. As we show in Section 5, assuming that column headers appear in the first row of a table disregards approximately 32% of tables, which are more complicated, in a collection of annotated HTML tables. Furthermore, this assumption is invalid for an alarming 75% of spreadsheets. In other words, the WebTables approach only applies to a fraction of all data tables (68% and 25% of HTML tables and spreadsheets, respectively), due to its assumptions of simplicity for the structure of the data being processed.

From our analysis of a large number of downloaded spreadsheets and HTML tables, we observed that headers, data, and other row types are frequently intermixed throughout a single data table. As a result, our approach is based on a collection of *row classes* that we have created to describe the function of rows in nearly all true relational data tables that we have encountered. For instance, one row class represents data rows with the label "D", while the label "H" represents a header row. Based on a large collection of hand-annotated data tables that we use as a training set, we use supervised classification in the form of a CRF to estimate the correlation between row features and row labels, and also to determine the relative likelihoods of transitioning from one row class to another. These estimates are used by the CRF to assign labels to rows outside of our training set.

Processing individual tables with our CRF-based classifier allows us to determine which tables contain truly relational data and to extract schema information by utilizing assigned row classifications. The first distinction is necessary because people use spreadsheets and HTML tables for many creative purposes that do not communicate structured data. For example, spreadsheets can be used as forms or instruction

manuals, while HTML tables are used far more often for controlling page layout than for actual data representation. We consider these tables to be *non-relational*, since they do not contain records of consistent types, and they are not useful to our schema extraction endeavor. As a result, given an input document, our technique can be applied to all data tables, but low scoring tables are treated as non-relational and discarded.

After determining that a table is relational, the row classes assigned by our classifier can be used to augment the original table with a more strictly relational structure, which can then be processed in a variety of ways. The most obvious approach is to concentrate on the data by simply discarding specific types of rows that are not relevant. In this way, our procedure can be used as a pre-processing phase to maximize the pool of usable tables that are accessible to downstream applications. This is the primary motivation for our method, as it allows the handling of tables that are outside the scope of the WebTables method. Of course, the results of our method are useful beyond serving as a simple filter for data rows. For example, when a table of city-level statistics is partitioned by state and this is indicated by a "group header row" before each partition, the state names will be lost if data rows are treated individually. This is analogous to database normalization, in a sense, because the state name appears only once, yet it applies to many data rows. Thus, to output a more complete table, one can "denormalize" this data by appending a column to the resulting table that contains the state name. The conversion of tables to a fully relational structure is not the primary focus of this work, but should be an interesting area to explore in the future.

The schema extraction process is visualized in Figure 1. In the rest of this section, we introduce data table terminology and concepts, define row classes that we attempt to identify within each table, describe the row features that we use as input for our classifier along with the details of the classifier itself, and finally outline how the output of the classifier can be used to interpret the data tables.

3.1 Preliminaries

Before we describe our method for identifying and extracting the layout of the tabular data, referred to as its *schema*, we briefly describe some concepts used in our schema extraction methods. Documents (i.e., spreadsheets, HTML pages) containing tabular data are divided into *data tables*, which are grids of table cells whose schemas are yet to be determined. These typically correspond to individual worksheets of a spreadsheet or individual HTML tables in an HTML page. Each relational data table has a *schema*, which, in our context, consists of attribute names, values, and types, where attribute names are column titles, attribute types are the types of values in the column, and attribute values correspond to data values in the column's cells. Column names are stored in a special row or rows, usually near the head of the table, called *header rows*, while the data is stored in rows referred to as *data rows*. The data table may also contain descriptions of data, which we refer to as *metadata*. Metadata appears for several reasons, such as table titles, notes about the original source of the data, and aggregations such as column subtotals. Some metadata rows are valuable in interpreting the table's data, such as when table data is grouped and a row with a single value serves as a group header, describing a category for the following rows. The goal of our schema extraction process is to identify valid data tables along with their associated schemas and data values, which can be subsequently stored in a relational database or processed for a custom application.

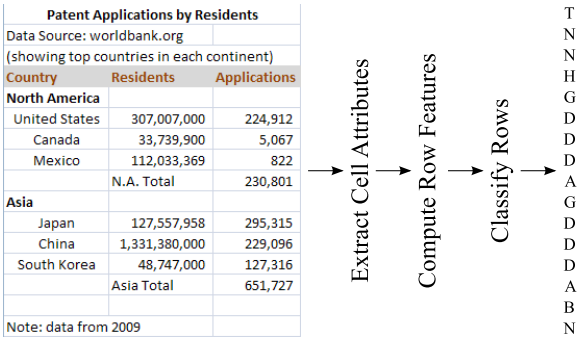


Figure 1: The row labeling process for a sample data table. The list of values on the right are the resulting row labels. In this case, the classifier correctly identified the first row as a title row, followed by two non-relational rows, followed by a header row, etc.

Both spreadsheets and HTML pages can contain tabular data. While spreadsheets consist entirely of data tables, HTML pages can contain additional content, as well as data tables in the form of *HTML tables*. HTML tables are defined by the <table> tag, and consist of a collection of rows and cells, denoted by <tr> and <th>/<td> tags, respectively. HTML tables have similar layout options to spreadsheets, including merged cells (through the use of rowspan and colspan attributes), header positioning, and metadata row inclusion. In addition, unlike spreadsheets, HTML tables support the inclusion of some additional structure relevant to our task, namely the <thead> and <th> tags, which serve to mark header regions and cells in tables. However, while the presence of these tags is a useful feature in determining the type of rows they occur in, they are often used for controlling layout and style, rather than for defining relational header rows in HTML tables. We also found that many HTML tables do not make use of these tags, as was also observed in prior work [18, 26]. An added complexity when processing HTML tables is the determination of which visual styles apply to the text in each cell. Our system for importing HTML documents incorporates Cascading Style Sheets to address this, including inline, embedded, and external style sheets, when evaluating the visual attributes of table cells. However, we observed that the external style sheets only yield modest improvements in accuracy, at the cost of slower processing speeds due to the increased number of resources that must be downloaded for each document.

Another consideration that must be made is how to handle tables that are not a simple n x m grid of table cells. Merged table cells (via the COLSPAN and ROWSPAN HTML attributes or a spreadsheet's "merge cells" feature) and nested HTML tables are two ways that this situation arises. Tables with either of these structures are accepted by our implementation. Our classifier includes features indicating the presence of merged cells, and after classification takes place, multiple copies of merged cell values are substituted for the original cell in header and data rows, so that each cell contains an appropriate string. Finally, unlike some previous work [24], we attempt to extract schemas from all HTML tables, including nested tables (i.e., not just leaf tables).

3.2 Row Classes

The vast majority of table rows serve functions that can be divided into a small set of *row classes* (each identified by a single-letter *row label*), which we describe here. Each represents the classification for individual rows of a data

table, as defined in Table 1. A minimal fraction of rows we observed (less than .01%) did not fit cleanly into one of these classes, such as rows containing notes beside cells that otherwise contain data values. In these cases, we assign the row label of the dominant class of the row.

Table 1: Row class definitions.

Label	Description
H	Header rows contain cell values that describe the values contained in the subsequent data rows of that column.
D	Data rows contain data records (or *tuples* in relational parlance).
T	Title rows describe the entire data collection found in the data table.
G	Group header rows provide categories for subsequent rows, for example a table containing demographic data about cities may be grouped by country.
A	Aggregate rows contain (typically numeric) summaries of preceding rows, such as totals/subtotals.
N	Non-relational metadata, such as a note, clarification, or any text that does not contribute data or structure to the data table.
B	Blank rows contain only empty cells.

3.3 Row Features

As with many machine learning tasks, a large factor in our classifier's accuracy is the quality of the input features. A difficulty with the current formulation of our extraction problem is that we desire a label for each **row**, yet each row consists of constituent **cells**, which can exhibit differing sets of attributes. Our approach to feature selection involves extracting a collection of attributes for individual cells, then combining attributes from all cells in the row using a novel binning scheme, in order to construct a set of row features. We first describe the cell attributes, followed by our method for combining them into row features.

3.3.1 Cell Attributes

We use a large battery of individual cell attributes as the basis for our features. These cell attributes fall into the following three broad categories, with specifics of each feature given at the end of this subsection.

- **Layout attributes.**

  Header rows often contain merged table cells with centered text, whereas data rows rarely contain merged cells and are typically right- or left-justified, so we make sure to include these properties as cell attributes. Additionally, for HTML tables, we incorporate the effects of tags like <thead> and <th> tag on the alignment of text within their cells (centered by default).

- **Style attributes.**

  Various font styles are more common in header rows or title rows than data rows, such as bold, italic, or underlined text. Differences in text and background colors and variations in date and number formats can also be used to differentiate between rows of distinct classes, so such attributes are extracted in our method.

- **Value attributes.**

  Header rows often contain relatively short textual values, rather than numbers or dates. While data rows may contain many empty cells, header rows typically contain few to zero blank cells across the full width of the table. Aggregate rows often indicate that the row contains a total or subtotal, so we include an attribute to match the case insensitive string TOTAL. All of these traits are extracted from data tables and used as cell attributes.

- **Similarity attributes.**

  Similarity attributes are formed as conjunctions of attributes of neighboring cells. For each cell c, neighboring cell c' (within the same column), and attribute α, we compute two similarity attributes for c, $c_{\alpha,A}$ and $c_{\alpha,B}$. We define $c_{\alpha,A} = c_\alpha \wedge c'_\alpha$ and $c_{\alpha,B} = c_\alpha \wedge \neg c'_\alpha$. These similarity attributes express similarities and differences between cells in neighboring rows, which are good indicators for row classes. (Note that for non-Boolean attribute α, $c_{\alpha,A} = 1$ iff $c_\alpha = c'_\alpha$ and $c_{\alpha,B}$ is not defined).

The full list of individual cell attributes is given below. Boolean attributes are prefixed with "Is" and end with "?". Other attributes take discrete textual values. The short text and long text attributes are given to text cells that contain text shorter or longer than one standard deviation from mean cell text length in a sample of data tables. Similarity attributes are conjunctions of individual cell attributes across multiple rows and are not listed below.

Table 2: Cell Attributes by Type

Layout	Style	Value
IsMerged?	IsBold?	IsEmpty?
Alignment	IsItalic?	IsText?
	IsUnderlined?	IsNumeric?
	IsColored?	IsDate?
	Font	IsShortText?
	Format	IsLongText?
		IsTotal?

3.3.2 Feature Binning

As with any machine learning based approach, the goal of the training process is to generalize the training data so that new, previously unseen data can be correctly classified. In the case of classifying rows in data tables, we encounter additional challenges to achieve high generalization of our classifier. As an example, upon encountering a 10-column spreadsheet where one of the rows contains 7 columns with centered text, a classifier should make an informed decision on how to label this row based on rows from a training set that share similar characteristics. In this case, it is important that other 10-column rows with 7 centered columns share a feature with this row, but we may also want an 11-column row with 8 centered columns to help inform the classifier's decision. Classifiers achieve highest accuracy when fed discriminative features, so we must decide on a feature encoding that properly respects these similarities.

One way to encode cell attributes into row features is with a *linear encoding*. That is, the percentage of cells in a row that share a cell attribute is used as a real-valued (i.e., non-Boolean) feature, with a value between 0 and 1. Linear feature encodings are used in several previous approaches to data table extraction, including the "CRF Continuous" method described by Pinto et al. [19]. However, using percentages for feature values can have some drawbacks. For instance, this approach treats a row with one numeric column as being more similar to a row with zero numeric columns, in terms of the distribution of row classes, than a row with

Table 3: Example feature encodings for raw feature value "3 of 11 row cells are numeric". The linear encoding method uses real-valued features, while the other methods use Boolean features.

Encoding	Feature Name	Value
Linear	"Percent numeric"	.27
Threshold	"< 50% of cells are numeric"	1
Direct	"3 of 11 cells are numeric"	1
Logarithmic Binning	"2 to 3 cells are numeric in a row of 8 to 15 cells"	1

three numeric columns, which we observe to be incorrect. Indeed, the distribution of row classes is unlikely to change linearly in step with the fraction of row cells that exhibit a specific attribute.

An alternative is to use a *threshold encoding*, which converts these fractional values into Boolean features. This encoding is used in the "CRF Binary" method presented by Pinto et al. [19]. Specifically, their system sets thresholds for various percentage features, such as an "Alphabet Characters" feature which is considered true if more than 60% of characters in a line are letters from the alphabet. The choice of threshold values for their system appears to be ad hoc for each feature, not systematic. More importantly, a single percentage-based threshold ignores differences between uniform and non-uniform rows.

Both the linear and threshold encodings can be problematic when used universally across tables with varying widths. "Narrow" tables with few columns exhibit different characteristics from "wide" tables with many columns, and should be treated differently when encoding features. That is, the cardinality of the columns exhibiting a specific attribute and the total number of columns in a row both matter, not just their ratio. For example, when 33% of a row's cells share an attribute in a 3-column table, the distribution of row labels can differ greatly from rows where 33% of a row's cells share the same attribute within a 15-column table.

To account for the drawbacks of linear and threshold encodings, we could use a *direct encoding*, which assigns a unique feature to each unique combination of (c, r), where c is the number of cells exhibiting an attribute and r is the number of cells in the row. However, this encoding greatly increases the number of possible row features and thus reduces the chances of each feature occurring in the tables of the training set, which will significantly reduce row classification accuracy, especially on wide tables.

We address the drawbacks of linear, threshold, and direct encodings by introducing a new "binning" scheme. A binning scheme partitions the space of possible raw feature values into bins, where each bin is assigned a representative feature value. The objectives of our binning scheme are given here.

- **Differentiate Between Table Widths**. Since tables with different widths generate different distributions of row labels, their features should be divided into separate bins.

- **Aggregate Wide Tables**. Tables with 2-6 columns are much more prevalent than wider tables. To account for the sparsity of some features on wider tables, bins for wider tables should also be larger.

- **Highlight Uniform Rows**. Rows where all cells either lack or share an attribute α should be in separate bins from non-uniform rows. Also, nearly-uniform rows should be in separate bins from more heterogeneous rows.

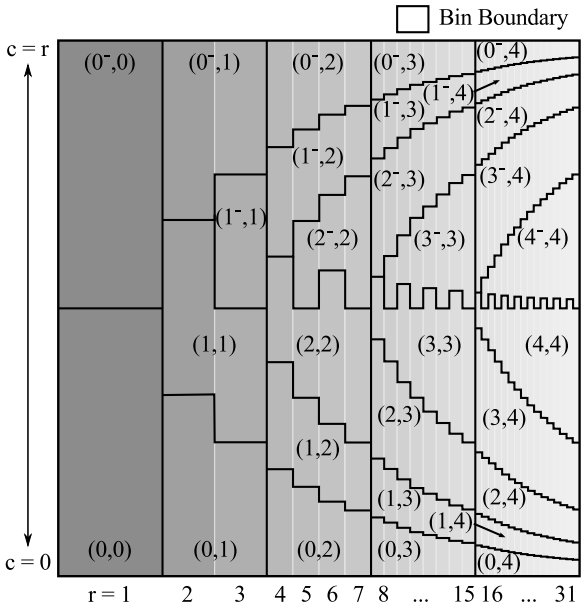


Figure 2: Graphical representation of the bin boundaries of the logarithmic binning encoding. For a given attribute α, two rows share a bin (and thus, have a common feature) if the base-2 logarithms of their r values are equal and the logarithms of their c values are equal (or $r - c$ values, if $c > r/2$).

These goals led to the encoding we present here, which we call *logarithmic binning*. The objective of logarithmic binning is to find an appropriate definition of "similar" in the table classification context. In the end, we say that two rows R_x and R_y are similar with respect to a certain cell attribute α if the logarithms of their widths are equal and the logarithms of the number of cells exhibiting or lacking attribute α are equal (all logarithms are base 2). This is implemented by assigning each row a representative feature for each attribute, based on these two log values. Formally, for row R_i of length r in which c cells exhibit a specific cell attribute α, we assign feature "$R^\alpha = (a, b)$" to R_i ((a, b) is denoted as its *bin*), where a and b are computed as follows.

$$a = \begin{cases} 0, & \text{if } c = 0 \\ \lfloor log_2(c) + 1 \rfloor, & \text{if } 0 < c \leq r/2 \\ \lfloor log_2(r - c) + 1 \rfloor^-, & \text{if } r/2 < c < r \\ 0^-, & \text{if } c = r \end{cases}$$

$$b = \lfloor log_2(r) \rfloor$$

The superscript minus sign in some values of a represents that it is computed based on $r - c$ rather than c.

Figure 2 provides a visual representation of the bin layout induced by logarithmic binning. In particular, it gives a visualization of the region of combinations of c and r values that are represented by each bin. The horizontal axis represents r and the vertical axis represents c/r. Horizontal partitions are made based on the logarithm of the number of columns in a row, while vertical partitions are made based on the number of cells within a row that exhibit (or lack) an attribute, given by the definition of a above. So R_1 from the previous example would fall into the bin for rows containing 4 to 7 cells ($b = 2$), of which 2 to 3 cells exhibit a specific

Table 4: Common table patterns.

HTML	Spreadsheet
HD	THD
THD	HD
HDA	TBHD
THDA	THDN
H(GD)*	HDN
H(BD)*	TBHD(BN)*

attribute ($a = 2$). R_2 would also fall into this bin, and thus the two rows share a feature under this binning scheme.

In the table extraction setting, the goal of logarithmic binning is to generalize the feature distributions better than the other encodings, and our experimental evaluation in Section 5 confirms that the schema extraction process benefits from its use.

3.4 Classifying with CRFs

After collecting the feature set for each row in a table, we can assign a row label to each row using a CRF-based classifier. As discussed in Section 2.2, CRFs are popular for sequence labeling in various domains such as natural language processing and computational biology.

We use CRFs in our system to assign row labels to each row in a new table. First, we collect a large collection of tables with human-annotated row labels to serve as a training set. The CRF is trained on this data using the limited-memory BFGS method [22] to determine optimal values for the model parameters. The logarithmic binning scheme results in a large number of possible features, however, the training time remains relatively low. One attribute of CRF classifiers that is beneficial given our choice of cell attributes is that CRFs can perform well even when multiple features are statistically correlated [22], as ours likely are. Using the CRF model parameters, dynamic programming is used to compute an optimal sequence of labels to assign to new data tables. This results in row label sequences, such as "TNNHGDDDAGDDDABN" for Figure 1, which we process in the following schema construction phase.

3.5 Schema Construction

The output of the CRF is a sequence of row labels which represents the row class assignments that are most likely according to the CRF model. Using the output of the CRF thus allows us to construct a relational schema for each data table. The header row(s) of the data table determine the column names, the type frequencies within the data rows of each column determine the data types, the group header rows determine additional attributes for each row that can be appended as a "Category" column, and the data rows themselves represent the data records that will populate the relational table. Depending on the application, titles and aggregations can also be saved as additional metadata describing the table, or as an indication that certain operations may be useful on certain columns.

4. COMMON TABLE PATTERNS

In this section, we present common table patterns. Using a large corpus of data tables with human-judged row labels, we analyzed the sequences of row labels for patterns (the corpus is described in detail in Section 5). We are interested in common table structures in the form of row label sequences.

Table 4 shows a listing of the six most common row label sequences for both HTML and spreadsheet tables. Consecutive instances of the same label are omitted to make the patterns more obvious. Patterns that involve repeated subsequences denote this with an asterisk. As expected, tables with one or more header rows followed by one or more data rows are very common in both table formats. Additionally, tables with a title row preceding the header and data are also common. Spreadsheets are more likely to contain blank or non-relational rows, so the other patterns are not common to both lists. These patterns are generalized and serve as the basis for an alternative method given and evaluated in Section 5.

Table 5: Dataset characteristics.

	Spreadsheets	HTML
Document count	14669	7883
Table count	46408	63009
Row count	5113070	215735
Unique domains	3636	4957

Domain suffix counts

Spreadsheets		HTML	
.gov	3476	.com	4835
.us	2829	.org	795
.uk	2710	.edu	576
.com	1592	.net	438
.org	1363	.gov	412
.edu	546	.uk	241

5. SCHEMA EXTRACTION EVALUATION

In this section, we present details of our evaluation of the schema extraction method as applied to datasets of spreadsheets and HTML tables from the Web.

5.1 Datasets

Since previous work has concentrated on extracting data from tables with simple structure, we found no previously created dataset that was adequate to test the accuracy of our method on complex tables. Instead, we created a new dataset of spreadsheets and HTML tables downloaded from the Web. In collecting our datasets, we sought a sample of tabular data from many different web sites and data sources so that we could test our methods' applicability across different data domains. To find relevant web sites containing tabular data, we performed several keyword searches, such as "spreadsheet" and "data table". For spreadsheets, we also explicitly searched for spreadsheet files with a .xls extension. These searches resulted in a list of Web sites, which we then crawled for spreadsheets as well as for pages containing HTML tables. Because spreadsheets and HTML pages often contain multiple tables, we consider them separately. Also, note that our current implementation accepts spreadsheets in .xls format, the default file format through Microsoft Excel 2003. The newer .xlsx file format is an ISO standard based on XML and serves as the default for newer versions of Microsoft Excel. However, based on several simple search engine queries at the time of writing, Web-accessible spreadsheets in the older .xls format appear to outnumber those in the newer .xlsx format by a ratio of roughly 50 to 1, meaning that we draw our corpus from a pool containing the vast majority of spreadsheets on the Web.

Table 5 lists some characteristics of our collected datasets. Both are of comparable size in terms of number of documents and number of tables. However, several differences are apparent. The HTML collection contains many more tables per document than the spreadsheet collection, due to HTML tables' frequent use for page layout, rather than for data presentation. Also, the total number of rows in the

spreadsheet collection far outnumbers the number of rows in the HTML collection, since spreadsheets tend to be used for data manipulation and aggregation, while HTML tables tended to be used for data presentation. Furthermore, analysis of the domain suffixes of data sources in each collection reveals that most spreadsheets were collected from government sources, while HTML tables were more frequently collected from commercial and organization domains. However, both collections have good source and content variety. Spreadsheets were sampled from 3636 distinct sources, and HTML tables from 4957 distinct sources. The data in these collections come from a broad range of websites, and span many different topics, including election results, product lists, radio stations, parking lot lat/long values, demographic data, and many others.

After creating our collections of spreadsheets and HTML tables, we randomly sampled a subset of these tables for manual annotation. An additional goal in the creation of our testing dataset was to exceed the size of hand-annotated table datasets used in previous work. The largest datasets found in work that we surveyed came from the WebTables paper and the work of Limaye et al. The former was tested on "a large sample of 1000 relations" judged by humans [2], while the latter was evaluated on collections of 437 hand-annotated tables, along with 6,085 machine-annotated tables [16]. Our experimental dataset size is far larger, as we hand-annotated 16,048 tables (2,259 from spreadsheets and 13,789 from HTML documents), of which 1,976 were relational (1,048 and 928, respectively) and required manual annotation of each row. We annotated each row of each table with the label corresponding to its row class: "H" for header, "D" for data, etc. Furthermore, we annotated the entire table as relational if it contained at least one header and one data row, and non-relational otherwise. Statistics from our annotations are listed in Table 6.

While the number of annotated documents is similar in both sets, there are considerable differences as well. Spreadsheet tables are much more frequently relational than HTML tables, consisting of 46% and 7% relational tables, respectively. This large difference is not overly surprising, since most HTML tables were used for page layout or other purposes. Furthermore, 7% relational HTML tables seems at a first glance quite small, but is comparable to the 1% relational HTML tables of WebTables [2]. The larger percentage is likely due to the targeted manner in which our datasets were created. Examining the number and types of annotated rows shows that spreadsheet tables tended to be far larger than HTML tables in the collection. We also studied the schema complexity of relational tables in our collections, in terms of several factors. First, we searched for tables with *simple schemas*—tables with a single header row followed by one or more data rows—and found that there was a much larger fraction of simple HTML tables than simple spreadsheet tables. We also found that more spreadsheet tables contained multiple header rows than HTML tables, and that more spreadsheet tables contained other (i.e., non-data, non-header) row classes than HTML tables. These three observations indicate that spreadsheet tables tend to have more complex structure than their HTML counterparts.

Note that the schema classifications as relational or non-relational are somewhat of a simplification, as in reality, examined tables were often not completely relational or non-relational. For example, we found a multitude of nearly-empty spreadsheet tables that were intended to be printed out and used as fill-in forms, which we classified as non-relational. Web-based calendars likewise use HTML tables for formatting, and have a quasi-relational structure but do not contain usable data. Additionally, some tables could be formatted as relations, but are not, such as tables that present street addresses using multiple table rows instead of multiple columns, and again we classified these as non-relational for our purposes.

Table 6: Tables annotated by human judges.

	Spreadsheets		HTML	
Annotated documents	1117		1204	
Annotated tables	2259		13789	
Relational tables	1048	(46%)	928	(7%)
Non-relational tables	1211	(54%)	12861	(93%)
Annotated rows	435160		20537	
Header rows	1479	(<1%)	978	(5%)
Data rows	425195	(98%)	18906	(92%)
Other row classes	8486	(2%)	653	(3%)
Relational tables:				
"Simple" schema	257/1048	(25%)	632/928	(68%)
Multiple header rows	157/1048	(15%)	63/928	(7%)
Other row classes	784/1048	(75%)	263/928	(28%)

5.2 Experimental Setup

We performed 10-fold cross validation on our collection of relational HTML and spreadsheet tables to obtain our classification results. The dataset was divided into 10 equally-sized groups, then 10 classifiers were trained on each subset of 9 groups, before running experiments on the remaining group. The reported results are averages from the 10 testing runs. The HTML and spreadsheet tests were performed separately in order to expose differences between the two table formats. Results were recorded for the following four separate classification methods.

- **WT**
  WebTables results were obtained using the "Header Detection" features and rule-based classifier from the Weka toolkit as described in the original WebTables paper [2]. Although the WebTables features for header detection are only defined for the first row, the extension to cover other rows in a table or spreadsheet can be performed by recomputing the WebTables features for rows after the first. An additional contribution of our work is the adaptation of the WebTables classifier to work with spreadsheets along with the HTML tables that they were originally designed to process.

- **B+A**
  The "Bayes + Automaton" method serves as a baseline method that incorporates global table structure. As in WebTables, a Bayesian classifier computes the estimated likelihood of each row being assigned each row label. But rather than choosing the most likely row label in isolation, a custom automaton is used to find the sequence of row labels with the highest aggregate likelihood that also adheres to the common table patterns discussed in Section 4, such as H(GD)*. For details on the automaton, see Appendix A. The full algorithm is given in Appendix B.

- **CRF-C**
  The CRF method with continuous features uses a linear feature encoding to form the input to a CRF. This is a similar encoding to the one used by the "CRF Continuous" method tested by Pinto et al. [19], but

uses the cell attributes and row classes we developed for spreadsheets and HTML tables.

- **CRF-B**
  The CRF method with binned features uses the full schema extraction method given in Section 3, including the use of the logarithmic binning from Section 3.3.2 before performing row classification with the trained CRF.

The cross-validation sets are partitioned by document (not by table) to prevent from using one table in a training partition when another similar table in the same document is in the testing partition. The reported accuracy, precision, and recall rates are averages over the 10 training and testing runs using each method.

5.3 Row Classification Evaluation

For our first evaluation, we tested the accuracy of the classifiers at the row classification task described in Section 3. This process yields the assignment of a row class to each row that represents our best guess of the row's purpose within the table (selected from our set of row classes in Section 3.2). The correct classification of table rows is a difficult task because a large number of tables in our corpus are not in a simple tabular format, such as those with one header row followed by a sequence of data rows. Instead, we found that many blocks are bordered by non-relational metadata, contain multi-row headers, or include subtotals or other noisy sections within a single schema block. However, the purpose of our row labeling procedure is to handle these cases, and our experiments show that it performs well.

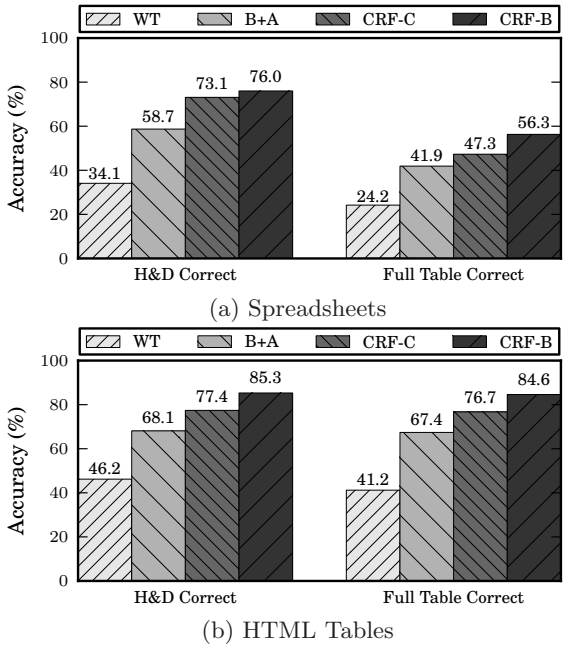
Table 7: Test set rows classified correctly.

	WT	B+A	CRF-C	CRF-B
Spreadsheets	97.6%	96.7%	99.3%	99.3%
HTML Tables	92.3%	92.7%	98.2%	98.1%

As shown in Table 7, the CRF-based methods obtain the best classification accuracy for both spreadsheets and HTML tables, ahead of WT and B+A. The higher scores on spreadsheet rows is a result of the higher proportion of data rows found in spreadsheets, as listed in Table 6. The row-level accuracy is an important metric for schema extraction. However, the gains are not fully illustrated when all methods have such high scores. If a hypothetical classifier were to label all rows as data rows (D), it would achieve a 97.7% accuracy rate on spreadsheets and 92.1% accuracy rate on HTML tables, since those are the fractions of rows in our test corpus that are data rows. Yet this classifier would be useless in the sense that all tables will contain misclassifications for non-data rows. We conclude that high row-level accuracy may hide the true power of these methods so we now look at full-table accuracy rates.

5.4 Full Table Accuracy

We evaluated the accuracy of the four classification methods on full data tables, because while individual row accuracy is important, errors anywhere within a document may will affect the usability of the data table in downstream applications. Correctly classifying individual rows is important. However, even a small number of incorrect row classifications can potentially lead to significant errors during schema extraction. Thus, we focused this experiment on the performance of our classifier on entire data tables. The goal, of course, is perfect classification of each row. However, the schema extraction applications we envision require



(a) Spreadsheets

(b) HTML Tables

Figure 3: Accuracy of classifiers on full spreadsheet tables and full HTML tables. The accuracy is measured as the percentage of tables in which correct labels are assigned to (i) all data and header rows, and (ii) all rows in the table.

highest accuracy for data rows, followed by header rows. If all of those are correctly classified, we can interpret the table relationally, even without correctly classifying the remaining rows (alternatively, we can treat them all as "non-relational").

The full table accuracy test measures the number of tables in which all rows are correctly classified and the number of tables in which all data and header rows are correctly classified (i.e., no false positives or false negatives for those classes). We make a distinction between these two cases because errors in header and data are critical to classify correctly for many purposes, while other row types may be "nice-to-have" for some applications, but are not always crucial. The results are shown in Figure 3. For both spreadsheets and HTML tables, the CRF-B method achieves the best accuracy for parsing full tables correctly. Even when only header and data rows are considered, which are what the WebTables features are designed to distinguish between, the WT method is significantly outperformed by the others.

It is also noteworthy that the document-level accuracy is higher for HTML tables, for all methods, despite the lower row-level accuracy in Table 7. This is likely due to the higher proportion of "simple tables" that exist in the HTML corpus, in combination with the lower overall proportion of rows that are data rows.

5.5 Effects of Feature Binning

The accuracy benefits of the CRF-based methods over WebTables and our automaton method are clear from the previous subsection. Now we examine the differences between the CRF methods, CRF-C and CRF-B in order to measure the effects of logarithmic binning. Since the only difference between the two methods is the feature encoding scheme, we can conclude that it is the sole cause of any changes in accuracy. The results of this test are displayed in Table 8. We use the standard precision and recall definitions, where precision is the proportion of all classifications

Row Class	Count	CRF-C Precision	CRF-C Recall	CRF-B Precision	CRF-B Recall	Change in F-Measure
Spreadsheets						
D	425376	.999	.999	.998	.998	−.001
H	1486	.937	.915	.945	.915	+.007
B	3792	.874	.862	.908	.974	+.071
T	702	.739	.756	.766	.822	+.046
G	1312	.669	.480	.758	.385	−.048
N	1877	.576	.709	.446	.639	−.111
A	615	.965	.703	.991	.890	+.123
HTML Tables						
D	18920	.988	.995	.991	.995	+.001
H	979	.921	.908	.911	.939	+.011
B	214	.852	.719	.984	.953	+.188
T	154	.702	.717	.875	.913	+.184
G	112	.667	.353	.545	.176	−.195
N	69	.667	.095	.120	.143	−.037
A	89	.059	.074	.706	.444	+.479

Table 8: The precision and recall for the CRF-C and CRF-B classification methods on spreadsheets and HTML tables. The change in F_1-measure that results from using the logarithmic binning scheme of the CRF-B method is also shown. Row classes are ordered by average frequency across table types.

of a specific row class that are true members of that row class (based on the human annotations), and recall is the fraction of all true members of a row class that were classified as such. The F_1-measure is computed as $(2 \cdot P \cdot R)/(P + R)$ for precision P and recall R, and the change in F_1-measure between CRF-C and CRF-B is displayed.

The results show some increases and some decreases in the F_1-measures for individual row classes. The high precision and recall of data rows is a result of their frequency in both spreadsheets and HTML tables. Both classification methods produce similar F_1-measures on these rows. At the opposite end, we note that the low precision and recall rates for group header rows, aggregate rows and non-relational rows in HTML tables is mainly due to the small number of tables containing rows of these types—which is the case because HTML table authors have the ability to place notes or explanations or other metadata within the document, but outside of the table. In contrast, all visible text in a spreadsheet is placed in spreadsheet cells, so there is no other place for notes to appear.

Although the use of logarithmic binning does not universally increase the F_1-measure across all row classes, we emphasize that it does achieve our main objective of increased full table accuracy. The primary value of these precision/recall results is to examine how row-level accuracy is affected by the use different feature encoding methods. To that end, we see that for both table types, CRF-B improves on CRF-C for blank rows, header rows, title rows, and aggregate rows, while the F_1-measure is reduced in the cases of group header rows and non-relational rows. The logarithmic binning scheme was not designed for the recognition of specific row classes, so the fact that the same row classes show improvements and reductions in accuracy for both spreadsheets and HTML tables is somewhat surprising. A common trait of both non-relational rows and group header rows is that they often contain a single non-blank cell in the leftmost column of a table, while the other cells are blank. Consequently, the divisions of cell features based on the number of total cells in a row may impede feature generalization and reduce the accuracy for these row classes. Examination of more complex binning schemes that address this is left as future work.

5.6 Row Class Ambiguity

One way to identify aspects of our approach that may need improvement is to examine the number of rows of each class that are confused for rows of each other class. We do this using the confusion matrix shown in Tables 10 and 9. Each cell of the matrix shows the percentage of all classified rows that were actually of the class with the label shown in the leftmost column, but were assigned the row label shown in the top row by the CRF-B classifier. The shaded cells along the diagonal show correct row classifications, while the surrounding cells show incorrect classifications. Row classes are ordered by average frequency across table types. Totals for each row and column are also displayed.

Table 9: Confusion matrix for CRF-B on spreadsheets.

Row label (true)	Row label (assigned) D	H	B	T	G	N	A	Row Sum
D	97.54%	0.00%	0.04%	0.00%	0.01%	0.16%	0.00%	97.75%
H	0.02%	0.31%	0.00%	0.00%	0.01%	0.00%	0.00%	0.34%
B	0.01%	0.00%	0.84%	0.00%	0.00%	0.01%	0.00%	0.87%
T	0.00%	0.00%	0.00%	0.13%	0.00%	0.03%	0.00%	0.16%
G	0.04%	0.00%	0.00%	0.00%	0.12%	0.14%	0.00%	0.30%
N	0.05%	0.01%	0.04%	0.04%	0.02%	0.28%	0.00%	0.43%
A	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.13%	0.14%
Col Sum	97.66%	0.33%	0.93%	0.18%	0.15%	0.62%	0.13%	

Table 10: Confusion matrix for CRF-B on HTML tables.

Row label (true)	Row label (assigned) D	H	B	T	G	N	A	Row Sum
D	91.64%	0.34%	0.02%	0.00%	0.02%	0.03%	0.08%	92.12%
H	0.24%	4.47%	0.00%	0.03%	0.02%	0.00%	0.00%	4.76%
B	0.05%	0.00%	0.99%	0.00%	0.00%	0.00%	0.00%	1.04%
T	0.00%	0.05%	0.00%	0.68%	0.02%	0.00%	0.00%	0.75%
G	0.08%	0.02%	0.00%	0.03%	0.10%	0.32%	0.00%	0.55%
N	0.19%	0.03%	0.00%	0.03%	0.03%	0.05%	0.00%	0.34%
A	0.24%	0.00%	0.00%	0.00%	0.00%	0.00%	0.19%	0.44%
Col Sum	92.45%	4.91%	1.00%	0.78%	0.18%	0.41%	0.28%	

An error-free classifier would result in zeroes for all of the non-shaded values (i.e., off of the diagonal). However, since our method does result in some misclassified rows, we can see where the errors lie. From the confusion matrix for spreadsheets, we can see that two of the non-diagonal cells have relatively high values. These are the cases of (D, N) (where the true row class is D, but the row class assigned by CRF-B is N) and (G, N). The fact that misclassified rows are often interpreted to be non-relational (N) is not surprising, since non-relational rows are fairly heterogeneous (especially in spreadsheets), and come in a variety of forms. That they are mistaken for data (D) and group header (G) rows is also not surprising, given the overall prevalence of data rows, and the frequent similarity between non-relational rows (such as notes) and group header rows, which both appear commonly with a single value in the first column of a row. True non-relational rows are also frequently misclassified.

The confusion matrix for classifying rows in HTML tables shows that the largest errors arise for (D, H), (G, N), (H, D), (A, D), and (N, D). These HTML row misclassification rates are all higher than their counterparts in spreadsheet rows. This has multiple causes, but is primarily due to the size differences between average HTML tables and average tables found in spreadsheets. Second, HTML tables are

generally narrower, which results in less contrast between the row types and higher levels of confusion. Finally, although the relative proportion of some row classes (i.e., G, A, and N) is higher in the HTML tables of our corpus than in the spreadsheet tables, the absolute number of rows of these classes is less, so there are fewer training examples. Many of the pairs of confused row labels for HTML tables are similar to those for spreadsheets. Additionally, both data rows and header rows are confused for each other, which can be attributed to the higher proportion of header rows in HTML tables. The errors on aggregate rows could potentially be reduced by incorporating more text indicators into our feature set, including words like "Average" and "Sum", which we observed in aggregate rows in our corpus of data tables.

5.7 Application to Existing Table Dataset

Our final experiment illustrates the accuracy of the CRF-B method on one of the largest pre-existing published table datasets, from the work of Limaye et al. [16] (we found no such dataset for spreadsheets). The Limaye dataset contains four types of relations, three of which are full HTML tables containing headers (the "Wiki_Manual", "Web_Relations", and "Wiki_Link" collections). The vast majority of these exhibit simple table structure and contain only header and data rows. Since the original classification goals of this dataset were different from ours, the existing annotations were not sufficient for our purposes, so we annotated each table in the dataset by hand for our row classification task. The statistics for each table collection are shown in Table 11. While Wiki_Manual and Web_Relations are small, Wiki_Link is very large, comprising nearly 6,000 relational tables. Tables with simple schemas make up a large percentage of each collection, which is expected since a method similar to WebTables was used to select them.

Table 11: Limaye dataset - table information.

Collection	# Relational	# Rows	% Simple Schemas
Wiki_Manual	38	1423	86.8%
Web_Relations	28	1837	78.6%
Wiki_Link	5753	120765	98.0%

We trained the CRF-B classifier on the dataset that was described in Section 5.1, and then tested that classifier on the three collections of HTML tables. The results, listed in Table 12, display the percentage of all rows that were correctly classified, along with the percentage of tables in which all H and D rows were correct and the percentage of tables that were entirely correct. Two important observations are worth highlighting. First, each collection in the Limaye dataset contains a larger percentage of simple schemas than is found in our own corpus of HTML tables. This means that using our method to detect and process tables could potentially increase the pool of accessible tables for applications such as the one described by Limaye et al. Second, even for the tables that were included in this dataset, and are assumed to have simple structure, our method is valuable, since the full table accuracy numbers in Table 12 exceed the percentages of tables with simple schemas. This is because a non-trivial number of tables with aggregates, row groupings, and non-relational data rows are present in this dataset and are accurately detected by our CRF-B classifier, but are not dealt with by simpler methods.

As one example of the benefit of our approach, we examined a table in the Wiki_Link collection that contains data about population density in regions of Italy. There are columns for Region, Population, Area, and Density, and data rows for every Italian region, followed by an aggregate row for the same statistics for all of Italy, with the string "Italy" in the Region column. The final row is correctly detected as an aggregate row by our CRF-B method, due to its bold font formatting, which allows it to be treated differently from the other rows in the table. However, if the row is not segregated from the others, methods such as the Least common ancestor (LCA) method evaluated by Limaye et al. would be adversely affected when determining the column type and cell entity assignments. Even algorithms that are flexible enough to handle multiple types in a single column could benefit from having less noise from aggregates such as this example, or group headers and non-relational row text. Thus, the use of our method as a preprocessor could improve accuracy of table extraction methods like this by filtering rows that do not fit with the data rows in the table.

Table 12: Limaye dataset - results using CRF-B classifier.

Collection	Rows	Tables - H&D	Tables - Full
Wiki_Manual	99.9%	100%	97.4%
Web_Relations	99.1%	89.3%	89.3%
Wiki_Link	99.9%	99.1%	99.1%

6. CONCLUSIONS AND FUTURE WORK

Computers are not adept at creating structure from unstructured information, so they should preserve structure wherever it can be found. We first developed a set of row classes that represent the most common functions of individual rows in a data table. We then identified cell attributes that we combined using the novel logarithmic feature binning technique to serve as input to a classifier based on conditional random fields. The classifier outputs the sequence of row labels with maximum likelihood based on these inputs, thus determining the row classifications that are used for extracting data and structure from the table. This method was shown to lead to a significant improvement over the existing WebTables approach, and our logarithmic binning scheme shows improvements over alternative feature selection methods. Specifically, our CRF method with logarithmic binning showed substantial improvement over all alternatives in the full table extraction test, which demonstrates our method's ability to process arbitrary tables. Based on the increasing prevalence of schema matching techniques, and applications based on utilizing tabular data from the Web, it should be clear that improving table extraction accuracy and recognizing additional table structures is crucial to their future success. Furthermore, answering queries using a repository of spreadsheets and HTML tables is an exciting venue for future work, as it involves query processing over noisy tabular data. We intend to explore the application of our technique to systems for joining Web tables, focusing on the common case where geographic tables contain spatial attributes [10, 20, 21] using both textual [7] and spatial [11] join techniques.

7. REFERENCES

[1] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[2] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Uncovering the relational web. In *WebDB*, Vancouver, Canada, June 2008.

[3] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the power of tables on the web. In *VLDB*, pages 538–549, Auckland, New Zealand, Aug. 2008.

[4] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. In *VLDB*, pages 1090–1101, Lyon, France, Aug. 2009.

[5] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai. Mining tables from large scale HTML texts. In *COLING*, pages 166–172, Saarbrücken, Germany, July 2000.

[6] E. F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, June 1970.

[7] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817–828, Scottsdale, Arizona, USA, May 2012.
[8] D. W. Embley, M. Hurst, D. P. Lopresti, and G. Nagy. Table-processing paradigms: a research survey. *IJDAR*, 8(2):66–86, 2006.
[9] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *WWW*, pages 71–80, Banff, Canada, May 2007.
[10] G. S. Iwerks and H. Samet. The spatial spreadsheet. In *VISUAL*, pages 317–324, Amsterdam, The Netherlands, June 1999.
[11] E. Jacox and H. Samet. Spatial join techniques. Computer Science Technical Report TR–4730, University of Maryland, College Park, MD, June 2005.
[12] D. Jannach, K. Shchekotykhin, and G. Friedrich. Automated ontology instantiation from tabular web sources—the AllRight system. *Web Semantics*, 7(3):136–153, Sept. 2009.
[13] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, Williamstown, Massachussetts, USA, 2001.
[14] O. Lassila. The resource description framework. *IEEE Intelligent Systems*, 15(6):67–69, 2000.
[15] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. Spatio-textual spreadsheets: Geotagging via spatial coherence. In *SIGSPATIAL*, pages 524–527, Seattle, WA, Nov. 2009.
[16] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1):1338–1347, 2010.
[17] 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, Vancouver, Canada, June 2007.
[18] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. In *VLDB*, pages 908–919, Istanbul, Turkey, Aug. 2012.
[19] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *SIGIR*, pages 235–242, 2003.
[20] H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17(6):647–656, November/December 1984.
[21] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *CACM*, 46(1):63–66, Jan. 2003.
[22] F. Sha and F. C. N. Pereira. Shallow parsing with conditional random fields. In *HLT-NAACL*, pages 213–220, 2003.
[23] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538, June 2011.
[24] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *WWW*, pages 242–250, Honolulu, HI, May 2002.
[25] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108, Scottsdale, Arizona, USA, May 2012.
[26] R. Zanibbi, D. Blostein, and J. R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. *IJDAR*, 7(1):1–16, Mar. 2004.

APPENDIX

A. AUTOMATON CONSTRUCTION

Here we describe the nondeterministic finite automaton (NFA) used in the B+A row classification method in Section 5. Figure 4 shows a portion of the automaton that recognizes tables that have one or more title rows (T), followed by one or more header rows (H), followed by one or more "data blocks". A "data block" consists of group header rows (G), data rows (D), and aggregate rows (A), and since we expect data blocks within the same table to observe the same structure, each valid combination of these provides a separate path in the NFA. The remainder of the graph is similar, but allows header rows (H) to be repeated instead of appearing prior to all data.

In the given portion of the automaton, we see that the label sequence "THDDDA" would be accepted, because there exists a path with consecutive edges marked with those labels which starts in the START state and ends in an ACCEPT state. The path includes self edges (not shown), with label D, and finishes in the second ACCEPT state from the right.

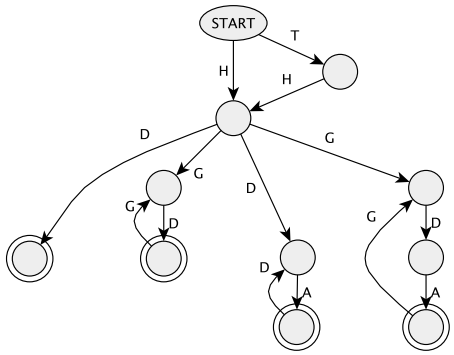


Figure 4: A portion of the automaton used for the B+A classification method. ACCEPT states are marked with a double border. Three self-edges exist for each node, but are hidden to reduce clutter. These have edge labels for blank rows (B), non-relational rows (N), and rows with the same label as the other incoming edge(s) to the node.

B. B+A ALGORITHM

Algorithm 1 shows the row classification procedure for the B+A method of Section 5. Before this algorithm is run, a Bayesian classifier is conditioned on a training set of data tables with human-judged row labels. To classify a new data table, the CLASSIFYROWSB+A function is invoked with parameters consisting of the row features of each row (R), and the total number of rows (n). Estimated probabilities of each row i being assigned each label l are obtained in lines 4-5, using the results of the Bayesian classifier (returned by GETBAYESROWPROBS) on the row features of each row individually ($R[i]$). In lines 8-18, the probability of the most likely sequence to end in each state s after each row i is determined. The intermediate results are stored in two arrays, $BEST[i, s]$ to store the best (maximum) probability of any sequence to reach state s after row i, and $LABELS[i, s]$ to store the sequence of states that obtains that probability. Finally, in lines 19-24, the most likely sequence that ends in one of the ACCEPT states (S_{ACCEPT}) is identified and returned.

Algorithm 1 B+A row classification algorithm

```
1: S ← { states of automaton }
2: L ← { set of row labels/automaton edge labels }
3: function ClassifyRowsB+A(R, n)
4:   for i ← 1 to n do
5:     P_i ← GetBayesRowProbs(R[i])
6:   Initialize 2D array BEST to 0.0.
7:   Initialize 2D array LABELS to [] (empty sequence).
8:   BEST[0, 0] ← 1.0 (initialize start state)
9:   for i ← 1 to n do
10:    for s ∈ S do
11:      if BEST[i − 1, s] > 0 then
12:        for l ∈ L do
13:          S' ← {states reachable from s by label l}
14:          for s' ∈ S' do
15:            B ← BEST[i − 1, s] · P_i[l]
16:            if BEST[i, s'] < B then
17:              BEST[i, s'] ← B
18:              LABELS[i, s'] ← LABELS[i, s] + l
19:  maxprob ← 0
20:  for s ∈ S_Accept do
21:    if BEST[n, s] > maxprob then
22:      maxprob ← BEST[n, s]
23:      labelseq ← LABELS[n, s]
24:  return labelseq
```