Understanding Tables on the Web

ABSTRACT

The Web contains a wealth of information, and a key challenge is to make this information machine processable. Because natural language understanding at web scale remains difficult and costly at present, in this paper, we focus our attention on understanding well-structured html tables on the Web. From 0.3 billion Web documents, we obtain 1.95 billion tables, and 0.5-1% of these contain meaningful information of various entities and their properties. Our work focuses on detecting these tables, understanding their content, and using the obtained information and knowledge to support important applications such as search. Our starting point is a rich, general purpose taxonomy whose content is harvested automatically from the Web and search log data. We use the taxonomy to help us interpret and understand tables. We then use the content we understand to enrich the taxonomy, which, in turn, enables us to understand more tables. We report large scale experimental results that demonstrate the feasibility of this approach, and we build a semantic search engine over tables to demonstrate how structured data can empower information retrieval on the Web.

1. INTRODUCTION

The World Wide Web contains a wealth of information. Unfortunately, most of this information is understood only by humans but not by machines. A key challenge is thus to make such information machine accessible and processable.

In this paper, we focus on mining information from structured data that reside within Web documents. The particular structured data we are concerned with are tables. The reason we choose tables is two-fold. First, there are billions of tables on the Web, and many of them contain valuable information. Second, tables are already well structured and relatively easier to understand, whereas converting free text into structured data using natural language processing techniques is a very costly and time consuming process for large web corpus.

Consider the tables in Figure 1. Our goal is to "understand" these tables, so that we can use the information therein to empower a large variety of applications.

Barack Obama	Aug 4 1961	Illinois
Hillary Clinton	Oct 16 1947	New York

(a)

Illinois	Springfield	12,900,000
New York	Albany	19,490,000

(b)

Names	Date of Birth
Richard Nixon	Jan 9 1913
Hillary Clinton	Oct 16 1947

(c)

Figure 1: Three tables from the web

One immediate question is how to evaluate understanding? To answer this question, we build a semantic search engine for tables. For any keyword query, the semantic search engine returns tables or snippets of tables as answers. A better understanding of the table will enable the search engine to return results that are more relevant from user's point of view, just like in Web search. In our study, we deploy the search engine on two corpora: tables on the World Wide Web (0.3 billion Web pages, or over 10 Terabytes) and over 0.65 million tables in Wikipedia. Figure 2 shows a snapshot of this search engine in action. More screenshots can be found at our website [1].

We next illustrate the understanding of tables with three example queries.

For query "*Obama birthday*", the search engine returns [1]:

Politicians	Date of Birth
Barack Obama	Aug 4 1961

Clearly, the search engine derives semantics from the data. It figures out that Obama is a politician, and Aug 4 1961 is his birthday. Such information is not explicit in the data.

For query "*Illinois*", the semantic search engine returns:

State	Capital City	Population
Illinois	Springfield	12,900,000

Note that i) the result has semantics (a table header), and ii) though the word *Illinois* appears in Figure 1(a) as well, the search engine realizes Figure 1(a) is more about Obama than about Illinois, so it is not returned (or ranked lower).

For query "*American politicians*", the semantic search engine returns:

[1]Our semantic engine searches over tables and returns the result in the form of tables as well.

Figure 2: The snapshot of our system

Politicians	Date of Birth	State
Barack Obama	Aug 4 1961	Illinois
Hillary Clinton	Oct 26 1947	New York
Richard Nixon	Jan 9 1913	-

In this case, the semantic search engine figured out that all three are American politicians, although this information is not obvious in the original tables.

Unlike text in natural language, a table shows how data is organized, that is, it reveals the structure of the data. However, knowing the structure does not mean that we know the semantics, or that we understand its content.

For example, in Figure 1(a), the machine has no idea *Barack Obama* is a politician, *Aug 4 1961* is a date, *Illinois* is a country, nor does it know that *Aug 4 1961* may not be just any date, but very likely the date when Obama was born or when Obama died, as *date of birth* and *date of death* are two important pieces of information about a person (and a politician is a person). Furthermore, the machine does not know whether *(Barack Obama, Aug 4 1961, Illinois)* is more about Obama the politician or about Illinois the state.

How would a human being understand a table? For Figure 1(a), even if we know very little about Obama beyond recognizing the name, we know immediately that the table is about two politicians, and common sense tells us that the third column, which contains names of state, very likely represents the home state of the politicians, and the second column, which contains two dates, is very likely birthdays of the two politicians, although they can also be other dates. Figure 1(c) is easier to understand, as it comes with a header, so we know the semantics of the data immediately. Interestingly, it contains a same politician *Hillary Clinton* with the same date *Oct 26 1947*. Since the date is in the column of "Date of Birth", we are certain that the second column of the first table represents birthdays rather than other dates as well.

The question we want to answer in this paper is, can machines achieve the same level of understanding?

A human being uses common sense and knowledge of worldly facts (e.g., Obama is a politician, Illinois is a state, and birthday is an important date associated with a person) to understand a table. We need to furnish a machine with the same knowledge in order for it to understand tables.

For this purpose, we build Probase, a rich, general purpose taxonomy of worldly facts from a corpus of 0.3 billion Web pages and other data. One dramatic difference between Probase and any manually built taxonomy (e.g., Open Mind Common Sense [18], Freebase [7], etc.) is that Probase has over 2.7 million concepts, which contain most if not all concepts about worldly facts in human minds. Each concept contains a set of entities or instances ranked by their popularity and other scores, and also a set of attributes that are used to describe entities in that concept. These enable us to understand a large range of tables of all possible content.

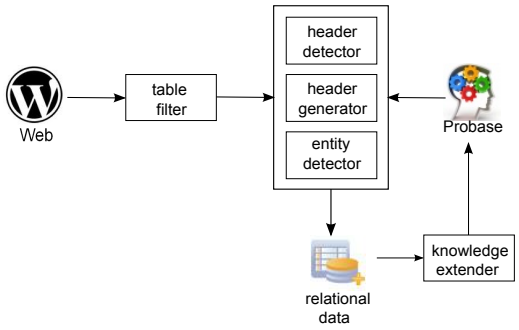

Figure 3: The flowchart for understanding tables

In this work, we first find tables that contain useful information (there are about 1.95 billion tables in a corpus of 0.3 billion web pages, and about 0.5% to 1% of them contain useful information). Then we analyze each table to find semantics of its content, which involves, e.g., detecting or generating a header for the table, identifying entities in the table, etc. Finally, we describe interesting applications, e.g., the semantic search engine we mentioned earlier, that can benefit from information mined from tables. Another important application is to use the information in the table to extend our knowledge about the world, i.e., we enrich Probase with entities (including attributes and values) and relationships we collect from tables.

Figure 3 shows the flowchart of the process. We first try to detect the header in a table. If no header can be detected, we generate one using the content of the table. Then, we identify the column that contain entities (other columns are attribute columns). If we fail to derive the header or the entity column, or the quality of the result is considered too low, we discard the table. Otherwise, we generate a set of statements from the table. These information can be used to enrich Probase.

The remainder of the paper is organized as follows. Section 2 describes the Probase taxonomy and its extension for understanding tables. Section 3 presents methods to find tables that may contain useful information. Section 4 discusses approaches to interpret the content of tables. Section 5 showcases two applications that leverage the table understanding. We evaluate our prototype system in Section 6, and discuss some of the most related work in Section 7 before concluding in Section 8.

2. TAXONOMY AND ITS EXTENSION

To understand a table, we must know its schema. The schema usually includes a set of attributes (e.g., *birthdate*, *party*, etc.) for a concept (e.g., *politicians*). Thus, it is important to know all the concepts, and most importantly, for each concept what are the attributes that describe the concept. Since tables from the Web are extremely diverse in topics and domains, we need a rich ontology that encompasses the concepts in the mind of all table creators.

One approach is to obtain concepts and attributes from trustworthy sources such as Wikipedia [4], DBPedia [5], and Freebase [7]. Wikipedia infobox contains high quality attributes for well known entities. For example, Wikipedia's page for Microsoft uses attributes such as *Founders*, *Headquarters*, *Revenue*, etc., to describe Microsoft. However, Wikipedia doesn't have explicitly listed attributes for each concept. One possibility is to propagate attributes from well known entities to other entities in the same concept. The problem is that the number of well known entities and the number of concepts in Wikipedia are very small, they do not have a good coverage of all kinds of entities and concepts out there. DBPedia [5] is an ontology manually constructed from Wikipedia's infobox, and currently it contains 259 classes and 1,200 distinct attributes. Freebase [7] has around 1,500 classes and 3,500 distinct attributes, while each class has 3.74 attributes in average. These suggest that obtaining attributes from existing manually created sources is not sufficient.

Another approach focuses on deriving concepts and attributes automatically. Pasca *et al.* [15] proposed an approach that finds attributes for a given concept. The idea is to manually create some "seed attributes" for the concept, then detect patterns that associate the seed attributes and the concept, and use the patterns to search the web to find other attributes. The approach however has scalability issues. First, it is infeasible to manually create "seed attributes" for as many as 2.7 million concepts in Probase. Second, even if we find all such "seed attributes," matching all the derived patterns against the entire web corpus to find other attributes is too time-consuming even on a large cluster. Another project, KnowItAll [12], focuses on extracting classes, instances and relationships from search engine's results by using Hearst's patterns. However the hierarchy thus built suffers both in size and the depth.

In this paper, we use the Probase [2] taxonomy to understand tables. Probase is a research prototype that aims at building a unified taxonomy of worldly facts from web data and search logs. The backbone of Probase is constructed using the Hearst linguistic patterns [13], or "SUCH AS" like patterns. For example, a sentence that contains "... politicians such as Barack Obama ..." can be considered as an evidence for the claim that *politicians* is a hypernym of *Barack Obama*.

Probase is unique in two aspects. First, the Probase taxonomy is extremely rich in concepts. The core taxonomy alone (which is learned from 0.3 billion web pages and 2 years' search log) contains more than 2 million concepts, while the well known Freebase [7] taxonomy contains about 1500 concepts. With 2 million concepts obtained directly from Web documents, the knowledge base has much better chance to include most if not all concepts in every human mind. Indeed, Probase has been very helpful to many applications where there is a need to interpret users' intention, for example, search applications [3],

Second, the Probase taxonomy is probabilistic, which means every claim in Probase is associated with some probabilities that model the claim's plausibility, ambiguity, or other characteristics. The probabilities can be derived from evidences found in Web data, search log data, and any other available data. Because of the probabilistic framework, it is natural for Probase to integrate information from other data sources, including other ontologies.

Despite their popularity in concept-entity extraction, Hearst patterns are not powerful enough for extracting attributes and values from web data. As a result, the core Probase taxonomy is not rich in the attribute/value space. To this end, we seek to enrich the Probase with more attributes, values (e.g., *Obama's birthday is Aug 4 1961*, and relationships (e.g., *Obama is a member of the Democratic Party*).

We use a two-phase approach to achieve this goal. In this section, we describe the first phase, in which we use a fixed pattern to find seed attributes for each concept. These seed attributes are good enough for detecting table schemata. Once we identify the schema, we can use the schema to enrich Probase by adding attributes, values, and relationships into the taxonomy. We describe the second phase in Section 5.2.

$\mathcal{C}, \mathcal{I}, \mathcal{A}$	concepts, instances, and attributes in the taxonomy
$E(c)$	entities of concept c
$A(c)$	attributes of concept c
$C(i)$	concepts entity i belongs to
$p_c(i)$	plausibility of entity i for class c
$a_c(i)$	ambiguity of entity i for class c

Table 1: Probase Notations

In our approach, we use the following linguistic pattern to discover seed attributes for a concept **C**.

What is the A of I?

Here, A is the seed attributes we want to discover, and I is an entity in concept **C** which is obtained from Probase. For example, assume we want to find seed attributes for concept *Country*. From Probase, we know *China* is a country. Then, we use pattern "What is the * of China?" to search the Web. A match "What is the population of China?" indicates *population* is a candidate seed attribute for concept *Country*. Below, we address several important issues in seed attributes discovery. The notations we use in our discussion are summarized in Table 2.

1. Why use the rigid pattern "What is the A of I"? We deliberately choose a rigid pattern to increase the precision. Because i) we are only interested in seed attributes in this phase, which will be expanded later; ii) we use multiple entities to find attributes for each concept, and iii) we are matching the pattern against a huge Web corpus (10 Terabytes), which means we can always find some good candidate attributes. Note that the above pattern is a skeleton which represents a set of concrete patterns. For example, the word *What* can be replaced by *Where* or *Who*, and *is* can be replaced by *are* or *was*. Thus, occurrences such as "Where is the birthplace of Barack Obama?" and "Who was the CEO of Microsoft?" indicate that *birthplace* and *CEO* are candidate seed attributes for concepts *persons* and *companies*, respectively.

2. What entities should we use as the **I** *in the pattern?* Probase [2] is a probabilistic taxonomy. For each concept, Probase returns a list of entities associated with a set of scores. Two essential scores are *plausibility* and *ambiguity*. Plausibility measures how likely an entity is really a member of a given concept, and ambiguity measures how likely an entity is also a member of other unrelated concepts. For example, both *Microsoft* and *Apple* are companies, but *Apple* is more ambiguous because it is also a fruit. For details on how the scores are derived, we refer readers to [2]. For our purpose, we want to use entities of high plausibility and low ambiguity (If we choose Apple as a company entity, we may mistake *color* as an attribute of company). Specifically, we find $ES(c)$, the eligible

entities of class c for the pattern.

$$ES(c) = \{i | i \in E(c), p_c(i) > \delta_p, a_c(i) < \delta_a\} \quad (1)$$

where $p_c(i)$ and $a_c(i)$ denote the plausibility and ambiguity of entity i with respect to class c, δ_p and δ_a denote the thresholds for plausibility and ambiguity, respectively.

3. How to rank candidate seed attributes to obtain final seeds? For each concept c, we score and merge candidate seed attributes derived from $ES(c)$ to obtain final ones for concept c. If attribute a is derived using the pattern with entity i, then we add a weight equals to $p_c(i)$ to a. We then aggregate the weights of all the candidate seed attributes, and choose top-ranked attributes as the final seed attributes of concept c. Specifically, let $D(a)$ denote the set of entities in $ES(c)$ having attribute a. The weight of a candidate attribute a with respect to class c is:

$$w_c(a) = \sum_{i \in D(a)} p_c(i) / \sum_{i \in ES(c)} p_c(i). \quad (2)$$

Note that Eq (2) considers not only plausibility but ambiguity as well, because $i \in ES(c)$.

One of the problem of using linguistic patterns to find attributes from the Web is that we may get a lot of noises, or wrong attributes. However, for different instances of I, the pattern typically returns different noises, hence the current ranking scheme effectively removes such noises.

Besides the above pattern, we also get attributes from DBPedia [5], which contains entities and their attributes in Wikipedia's infoboxes. We use exact matchings when locating Probase's entities in DBPedia.

In total, we obtain 10.5 million raw seed attributes (0.21 million distinct) for about 1 million classes. These are enough to bootstrap the process of understanding tables. Once we identify table schemata, they can be used to expand the current set of attributes. In an evaluation of 30 concepts and their top 20 attributes, we found our approach reaches an average precision of 0.96. This provides a strong foundation for further understanding process.

3. FINDING TABLES

From 0.3 billion web documents, we extract 1.95 billion raw html tables. Among these tables, many do not contain useful information or relational information (e.g., they are used for page layout purposes); others have structures that are too complicated for machines to understand. We use a rule-based table filtering method to acquire 65.5 million tables (3.4% of all the raw tables) that contain potentially useful information.

Before we describe our rule-based filtering method, we briefly introduce Cafarella *et al.*'s pioneering work [8, 9] of using machine learning to identify tables that are *relational* (that is, describing a set of entities with a set of attributes). First, it manually labels tables in a training set as relational or not relational; then it manually selects a set of features, for example, *number of rows*, *number of columns*, *average cell length*, etc.; finally it learns a classifier from those features.

Instead of using a supervised learning approach, we use a simple set of manually constructed rules to filter tables. The reason we choose rules over classifiers is the following: First, our goal in this step is to achieve high efficiency and high recall. We are not particularly interested in high precision. Thus, it is perfectly fine that some meaningless tables remain after our filtering, as they will have to go through another phase of semantic filtering (Section 5). In this case, a manually constructed filter is often as good as a learned classifier, since the classifier is derived from a small number of simple features (such as *number of rows/columns*). Second, since tables are so diverse, we may need a large training dataset in order to obtain a classifier of good accuracy, and the cost of manually labeling a large training dataset is very high. Thus, for the task of table filtering, using rules has the benefit of saving the cost of labeling without compromising the quality of the model.

Specifically, the simple set of rules we use for filtering tables are the following:

- Removing tables with too many rowspan/colspan cells: Tables with many "rowspan=" or "colspan=" constructs tend to have very complicated structures, and are often hard to understand. However, if there are only a few rowspan/colspan cells, we simply split them into multiple cells, so the table becomes two dimensional (and potentially relational).
- Removing tiny tables: By tiny tables, we mean tables whose dimensions are smaller than 2×2, or tables with 1 row or 1 column only. These tables are unlikely to contain useful information (they are more likely used for page layout), and even if they do, it is hard to extract the information correctly as no row-wise or column-wise statistics is available.
- Removing tables with too many hyperlinks: If most cells of a table contains nothing but hyperlinks, it is very likely that the table is for controlling page layout.
- Removing tables with too many empty cells: We discard a table if more than a certain percentage of its cells are empty. Currently, we set the threshold at 40%.
- Removing calenders: We found that quite a few tables that survive the above rules are calendars, and they are removed from consideration.

We pay special attention to tables on Wikipedia. These tables are created using a different syntax, so we need an additional parser to process them. Because Wikipedia has a much smaller scale, and tables on Wikipedia generally have higher quality, we also make some changes to the filtering rules we described above. First, we do not drop tables with many "rowspan" and "colspan". Instead, we always split a rowspan/colspan cell into multiple cells. Second, we do not remove tables with many hyperlinks, especially if the hyperlinks lead Wikipedia pages. Unlike filtering tables on the web, as many as 82% of tables on Wikipedia survive filtering.

4. UNDERSTANDING TABLES

In this section, we describe the process of *understanding* a table. With the help of Probase, we identify entities (of a single concept/class), attributes, and values in a table. Then, we use the information in the table to enrich Probase.

4.1 Knowledge APIs for Schema Extraction

We first introduce two functions $f(A)$ and $g(E)$, which are part of the knowledge API[2] provided by Probase. Probase contains rich information about concepts, entities, and attributes. For a given concept, we may use Probase to find its entities and its attributes. Likewise, for a given set of entities or attributes, we may use Probase to find the set of concepts that these entities or attributes may belong to. The two functions serve this purpose, that is, given a set of attributes A and a set of entities E, functions $f(A)$ and $g(E)$ find the most likely concept to which A or E belongs. More specifically, we have

[2]Real names of f and g are `findConceptByAttributes()` and `findConceptByEntities()` respectively. We use f and g for presentation simplicity.

- $f(A)$: for a set of attributes A, $f(A)$ returns a list of triples $\cdots, (c_i, A_i, sa_i), \cdots$ ordered by score sa_i, where c_i is a likely concept for A, $A_i \subseteq A$ are attributes that belong to concept c_i, and sa_i is the score indicating the confidence of c_i given A.

- $g(E)$: for a set of entities E, $g(E)$ returns a list of triples $\cdots, (c_i, E_i, se_i), \cdots$ ordered by score se_i, where c_i is a likely concept for E, $E_i \subseteq E$ are entities of concept c_i, and se_i is the score indicating the confidence of c_i given E.

The two functions may help us discover the schema of a table. Let us consider Table 2 as an example. Suppose we want to know if the first row is the head of the table. As we know, the head usually contains a set of attributes of a concept. So if we apply $f()$ on $A = \{$*Name, Birthdate, Political Party, Assumed Office, Height*$\}$, we may expect to get some concepts of high confidence. Assume Probase returns the following for $f(A)$:

(*US presidents, {Birthdate, Political Party, Assumed Office}, 0.90*)

(*politicians, {Birthdate, Political Party, Assumed Office}, 0.88*)

(*NBA players, {Birthdate, Height}, 0.65*)

...

The result ranks *US presidents* higher than *politicians* as the most likely concept. This is totally possible because we are only looking at the header of the table, and have not studied the content of the table yet.

On the other hand, if we compute $f(A)$ on the second row $A = \{$*Barack Obama, 4 Aug 1961, Democratic, 2009, 6'1*$\}$, we may get empty results or results with very low confidence. This indicates that $f()$ is useful in table header detection. We discuss this in more detail in Section 4.2.

However, many tables do not have a header. Is it still possible to understand such tables? Function $g()$ may help in this case. Consider the first column $E = \{$*Name, Barack Obama, Arnold Schwarzenegger, Hillary Clinton*$\}$ of Table 2. Applying $g()$ on E, we may get:

(*politicians, {Barack Obama, Arnold Schwarzenegger, Hillary Clinton}, 0.95}*)

(*actors, {Arnold Schwarzenegger}, 0.5}*)

...

In this case, $g()$ identifies *politicians* as the top match. If we apply $g()$ on other columns of the table, we may recover possible attributes for the concept. Thus, there is a possibility we can generate a header for the table. We discuss this in more detail in Section 4.3.

In summary, combining the outcomes of $f()$ and $g()$, we may get a clue regarding what the table is about. The scores sa_i and se_i returned by $f()$ and $g()$ play an important role in reaching the final conclusion. One important issue is how sa_i and se_i are calculated. Intuitively, the more matching attributes/entities we find, the higher score of a concept should be. Specifically, assume (c_i, E_i, se_i) is in the output of $g(E)$, and (c_i, A_i, sa_i) is in the output of $f(A)$, respectively. A straightforward way to define se_i and sa_i is as follows.

$$se_i = \frac{1}{|E|} \sum_{e_i \in E_i} p_{c_i}(e_i) \tag{3}$$

$$sa_i = \sum_{a_i \in A_i} w_{c_i}(a_i) \tag{4}$$

where $p_{c_i}(e_i)$ is the plausibility score of e_i given c_i, and $w_{c_i}(a_i)$ is the confidence score of a_i being the attribute of c_i in Probase.

Due to space constraint, in this paper, we omit the discussion of how the two functions are implemented in Probase, and refer the readers to [2] for details.

4.2 Header Detector

The first step in understanding a table is to locate the header. The header of a table contains a set of attributes, which form the schema of the table. The header also indicates the orientation of a table, depending on whether it appears as a row or a column. In our discussion, we assume all tables are horizontal, e.g., in Table 2, the header is the top row.

Let P denote the set of possible headers for a table T. In one extreme, we may consider every row of T, that is, P contains all rows of T. In the other extreme, we may just consider the top row, which is the mostly header if T has one. We use function $f()$ to evaluate each possibility and generate a set of candidate schema.

$$candidate\_schema = \{x \quad | \quad p \in P, x \in f(p), \\ x.sa + \alpha(p, T) > \gamma_h\} \tag{5}$$

Eq 5 filters out candidates whose score is below a threshold γ_h. The score consists of two parts, the raw sa score returned by $f()$, and an adjustment $\alpha(p, T)$, which evaluates whether a row is likely a header based on the syntax. The reason we need an adjustment is that the header usually has some syntactic characteristics that set it apart from the rest of the table. To name just a few:

- HTML <th> tag. The <th> tag is a good indicator of header cells.

- Other syntactic clues, including i) cells of p end with a colon ':', and ii) cells of p has a formatting (e.g., **bold** font is used) different from other rows in table T, etc.

- Data type differences[3]: The fact that a cell in p is of different data type from other cells in its column may indicate p is a header. For example, a cell in p has string type (e.g., *Birthdate* or *Age*), while other cells in that column are dates (*4 Aug 1961*) or numbers (e.g., *61*).

If $candidate\_schema$ returned by Eq 5 is empty, then we fail to detect any header. This is possible because tables may not come with a header. In this case, we generate a header using the method elaborated in Section 4.3.

As for the example of Table 2, a properly set threshold will find the first row as the header, with two candidate schema that are shown below. For simplicity, we assume $\alpha(p, T) = 0$ here, that is, we assume the table provides no syntactic clues for header identification.

(*US presidents, {Birthdate, Political Party, Assumed Office}, 0.90*)

(*politicians, {Birthdate, Political Party, Assumed Office}, 0.88*)

In Section 4.4 we will discuss how to further narrow the above two candidates down to a single interpretation.

4.3 Header Generator

If no candidate header is found in the header detection phase, then we assume that the table does not come with a header. In this case, we try to generate a header for the table.

For each column L_i, we find its most likely concept. For example, from the content in the 3rd column of Table 2, we may derive

[3]Although data types are not an entirely syntactic issue, we included it here since they can be detected syntactically.

Name	Birthdate	Political Party	Assumed Office	Height
Barack Obama	4 Aug 1961	Democratic	2009	6'1
Arnold Schwarzenegger	30 Jul 1947	Republican	2003	6'2
Hillary Clinton	26 Oct 1947	Democratic	2009	5'8

Table 2: A running example for table understanding

Political Party. Then, we check if the concepts of all columns together describe some other concept.

Specifically, for each column L_i, we first find its best matched top-k candidate concepts by calling $g(L_i)$. We denote the top-k candidate concepts of a given column L_i as

$$c^1(L_i), \cdots, c^k(L_i)$$

Then, we generate P, a set of possible headers by selecting one candidate concept from each of the top-k candidate concepts, that is

$$P = \{(a_1, \ldots, a_i, \ldots, a_n) | a_i \in c^k(L_i)\}$$

where n is the number of columns of a given table.

Finally, we derive $candidate\_schema$ using P and Eq 5. Note that in this case, $\alpha(p, T)$ is 0, as the table does not have syntactic clues for generated headers.

If $candidate\_schema$ is again empty, which means the table does not come with a header and we cannot generate a header for it, then we drop the table from further consideration.

4.4 Entity Detector

The entity detector tries to accomplish two tasks. First, it detects the *entity column* of the table. For example, for Table 2, the entity column is the 1st column, instead of the 3rd column, as the table is about politicians instead of political parties. Second, previously, we have derived a set of candidate schema for the table, and we need to narrow them down to one final interpretation. The method presented here solve the two problems at the same time.

We make our judgment based on two kinds of evidence. First, the entity column should contain entities of the same concept, and we can derive the confidence of a concept for a given column. Second, the header should contain attributes that describe entities in the entity column, that is, the candidate schema we have derived should match the concept of the entity.

For each $s \in candidate\_schema$ returned by Eq 5, we enumerate every column *col* and compute its confidence score. Given a column *col*, we define

- E^{col}: the set of all cells in *col*, except for the one in the header that corresponds to schema s
- A^{col}: the set of all attributes in s (except the one in the current column)

We then apply functions $f()$ and $g()$ on E^{col} and A^{col} to obtain their possible semantics. Specifically, we have

- $SC_A = f(A^{col})$, which contains a list of (c_i, A_i^{col}, sa_i) triples ordered by score sa_i;
- $SC_E = g(E^{col})$, which contains a list of (c_i, E_i^{col}, se_i) triples ordered by score se_i.

If *col* is the entity column, we should be able to derive a concept c from it, and the concept should also be strongly supported by A_{col}. So the possibility of *col* being the entity column relies on the confidence of the concepts derived from it.

We join the two lists of SC_A and SC_E to find common concepts, and we record the corresponding score as a multiplication of sa_i and se_j.

$$h(s, col) = \max\{sa_i \cdot se_j \quad | \quad (c_i, A_i^{col}, sa_i) \in SC_A, (c_j, E_j^{col}, se_j) \in SC_E, c_i = c_j\}$$

Finally, the most possible interpretation and entity column are the ones that achieve the maximum score.

$$(final\ schema, entity\ column) = \underset{s,col}{\text{argmax}}\ h(s, col) \quad (6)$$

Using this approach, for the example in Table 2, we will reject the schema of *US presidents*, because although the concept *US presidents* has strong support from the attributes, it has low support from the column that contains the name Arnold Schwarzenegger. Thus, the final schema we find for the table is:

(*politicians, {Birthdate, Political Party, Assumed Office}, 0.88*)

5. APPLICATIONS

Once we unlock the wealth of information hidden in web tables, we can enable a set of new applications. In this section, we introduce two of such applications: *A Semantic Search Engine over Tables* and *Taxonomy Expansion using Information in Web Tables*.

5.1 A Semantic Search Engine over Tables

To be exact, the semantic search engine we build does not operate upon tables, but upon table statements. A statement consists of a single entity, its attributes, and corresponding attribute values. Usually, a statement corresponds to a row of a table. For example, there are three statements (rows) in Table 2, each about one politician. A statement represents a basic unit of knowledge of an entity, and we use statements to answer a query. This differentiates us from previous work [9], which uses tables as the basic unit for answering queries.

Furthermore, instead of performing keyword matching in table search as in previous work, we try to find the semantics of a query, and return a set of statements that match the semantics. Specifically, we try to identify (at most) one concept c, (at most) one entity e, and possibly one or more attributes a (for the concept and/or entity we have identified) in a query. We treat everything else in the query as keywords k. For example, for query "American politicians birthday", we identify c ="American politicians" as a concept, a ="birthday" as an attribute of c. The process of identifying concepts, entities, and attributes is not trivial, as there may be exponential number of options. We refer the readers to [3] for a detailed discussion of query parsing.

Next, given a query represented by $\{c, e, a, k\}$, we decide what are the table statements that match the query. If a query specifies a concept c but does not specify entity e, then statements about any entity of concept c are potential answers (provided they also match attributes a, and keyword k). Table 3 lists the scope of the query based on its type, that is, we want to find all statements that are about any concept in C and any entity in E.

Query Type	C	E
entity only e	all concepts that contain e	$\{e\}$
concept only c	$\{c\}$	all entities of c
entity e + concept c	$\{c\}$	$\{e\}$
no entity or concept	all concepts	all entities

Table 3: Possible concept and entity set of a query

Finally, we need to score and rank all the statements that satisfy the query. Let s denote a statement, and let e', c', a', k' denote its entity, the class of the entity, attributes in its header, and its context (keywords) respectively. We consider two factors.

- Factor 1: How well do keywords and attributes match? We use

$$num_K(s) = |k' \cap k| \tag{7}$$

to denote the hits of keyword. Measuring how well two attributes match is more complicated, as we need to consider synonyms, for example, "birthday = date of birth". Furthermore, the meaning of a certain attribute must be considered within its concept. We consider a concept and an attribute that forms a relationship, for example, R(politicians, birthday), which means R contains all pairs of politicians and their birthdays, for example, (Barack Obama, 4 Aug 1961) $\in R$. Then, we use Jaccard similarity to define the similarity between two relationships R_1 and R_2, that is,

$$sim(R_1, R_2) = J(R_1, R_2) = \frac{|R_1 \cap R_2|}{|R_1 \cup R_2|}$$

Finally, we can define attribute match as

$$num_A(s) = \sum_{a_1 \in a} \max_{a_2 \in a'} sim(R_{c,a_1}, R_{c',a_2}) \tag{8}$$

where $R_{x,y}$ denotes the relationship formed by concept x and its attribute y.

- Factor 2: Does statement s have high quality? A high quality statement should have a plausible concept supported by both the header and the entity. This can be evaluated by the score returned by function $f()$ and the plausibility $p_e(e)$. We define $q(s)$ to be the quality of s:

$$q(s) = \max\{p_e(e') * sa(c) | c \in C\}, \tag{9}$$

where $sa(c)$ is the score of c in $f(a')$.

The ranking method takes both factors into consideration. Given a query q, we first find statements based on C and E. Then we sort candidate results by $num_A(s) + num_K(s)$ in descend order. If a tie happens, statements are ranked based on their own qualities $q(s)$. At last, a set of ranked statements are returned as the result.

5.2 Taxonomy Expansion

Probase enables better understanding of web tables. In turn, the knowledge derived from web tables can enrich Probase. In this section, we focus on how to expand entities, attributes and values, and relationships in the Probase taxonomy.

5.2.1 Entity Expansion

The backbone of the Probase taxonomy consists of hypernym-hyponym relationships extracted through Hearst's patterns from web data. Certainly, not every hypernym-hyponym relation is expressed in Hearst's patterns. Web tables are a good complementary source of information.

Expanding entities is straightforward. For example, based on Probase, we conclude that the leftmost column in Table 2 contains names of politicians, even if not every name there is found in the Probase taxonomy. The leftmost column provides an evidence that is equivalent to a Hearst's pattern "... *politicians such as Barack Obama, Arnold Schwarzenegger and Hillary Clinton* ..." Thus, we can expand Probase by incorporating this new evidence in the same way as we encounter a new Hearst's pattern.

Specifically, for each column col and its entity set E_{col}, we use $g(E_{col})$ to get highly representative concepts of it. If score se_i for a concept c_i reaches a threshold γ_t, then we believe c_i is good enough for generalizing the column. We thus insert a hypernym-hyponym relation (c_i, e_i) for every new $e_i \in E_{col}$ into the taxonomy with $p_{c_i}(e_i) = se_i$.

5.2.2 Attribute expansion

After the entity column is determined, we can expand the attribute list in Probase by adding unknown attributes to representative concepts of the entity column. This process can help us get more attributes in the light of tables, such as "Tel-#" for "phone number". Furthermore, enriching Probase by expanding attribute set and understanding more tables are in positive feedback loop.

When adding newly discovered concept-attribute pairs, a score $w_c(newa)$ is assigned to the new attribute $newa$ corresponding to concept c. There are two factors that can affect $w_c(newa)$. The first one is the confidence of concept c when given an entity column. We use the probability $p(c|E_{col}, A_{col})$ to represent it. The second one is the confidence of the new attribute to concept c. If we assume a new attribute have a similar status as other known attributes in A_{col}, then we can assign the score $sa(c_i)$ to $newa$, where $sa(c_i)$ is the score of c in $f(A_{col})$, because it denotes the average confidence of A_{col} to c_i. Then $w_c(newa)$ can be defined as follows:

$$w_c(newa) = p(c|E_{col}, A_{col}) * sa(c) \tag{10}$$

The remaining task is to compute $p(c|E_{col}, A_{col})$. Assuming E_{col} and A_{col} are independent, then

$$\begin{aligned} p(c|E_{col}, A_{col}) &= \frac{p(E_{col}|c) * p(A_{col}|c) * p(c)}{p(E_{col}) * p(A_{col})} \\ &= \frac{p(A_{col}|c) * p(c|E_{col})}{p(A_{col})}. \end{aligned} \tag{11}$$

Because $p(A_{col})$ is a normalizing constant, we only need to consider $p(A_{col}|c)$ and $p(c|E_{col})$. Intuitively, the prior probability $p(c|E_{col})$ indicates how possible the class c can generalize instance set E_{col}, so we let $p(c|E_{col}) = se(c)$. Because the weight $w_c(a)$ is derived from the percentage of entities having a in c, we can use it to indicate the possibility of a to be the attribute of c, i.e. $p(a|c) = w_c(a)$. Then we have $p(A_{col}|c) = \prod_{a \in A_{col}} p(a|c) = \prod_{a \in A_{col}} w_c(a)$. [4]

Now we have the value of $p(c|E_{col}, A_{col})$. If it reaches a threshold γ_a, we believe c is a credible concept and use the possibility to compute $w_c(newa)$.

6. EXPERIMENT

Since most of the evaluations are related to concepts, we randomly selected 30 concepts as our benchmarks, which vary in domains and sizes (See Table 4). For experiments that need to be

[4] $w_c(a) = 0$ means we have no evidence of $a \in A_{col}$, we assign a small value ϵ to $p(a|c)$ for smoothing in this case.

ID	Concept	E	New E	A	New A
1	academic institutions	431	31	482	42
2	actors	3466	101	26	36
3	airlines	1221	334	46	58
4	albums	1938	0	44	127
5	awards	4410	12	155	30
6	basketball players	22	0	19	26
7	beers	514	6	262	4
8	celebrities	8381	114	41	74
9	chemical elements	23	8	0	0
10	cities	9632	2940	360	216
11	companies	85391	416	53	643
12	countries	5779	5975	319	192
13	currencies	756	122	391	21
14	file formats	698	97	106	12
15	films	13402	116	40	367
16	football clubs	70	25	153	14
17	fruits	1631	20	121	0
18	guitarists	492	3	32	8
19	holidays	915	189	92	59
20	infectious diseases	879	6	194	0
21	newspapers	1630	86	47	18
22	operating systems	86	24	132	1
23	planets	335	54	592	51
24	politicians	953	53	109	33
25	programming languages	520	112	122	19
26	religions	1115	47	593	12
27	scientists	2677	1	56	16
28	songs	15372	31	59	490
29	sports leagues	75	3	407	2
30	video games	620	9	413	113

Table 4: 30 randomly selected concepts with high frequency (E = entities, A = attributes)

evaluated manually, we use a consistent scoring criterion for human judges: 1 for correct, 0.5 for partially correct, 0 for incorrect. Because of the space limitation, only some statistics of the results will be provided. Readers are referred to our website [1] for complete experimental results.

6.1 The Web Table Corpus

We implemented the prototype system with a Map-Reduce distributed computing platform. Each of the following stages takes several hours to complete.

1. Table filtering:

We ran filtering procedure the 1.95 billion raw tables extracted from 10 terabytes or 0.3 billion web documents, according to Section 3. The following observations can be made from the results. First, the percentage of tables with "rowspan" or "colspan" is not large (14.5%), and our samples of such tables show that only 10% of these are regarded "relational" by human, hence discarding this type of tables does not effect the final recall. Second, 81.4% of the tables that do not contain useful information are "tiny tables", which are often used for page layout. Third, 69 million tables out of 1.95 billion raw tables (3.5%) survived the filtering.

We then asked a human judge to label randomly sample of 200 filtered tables as either relational or non-relational. It turns out that 159 or 79.5% of samples are relational, which is an acceptable precision.

2. Header detection: We randomly selected sample of 200 tables from the filtered tables, and ran the header detection algorithm on them. The result is shown below.

Actual no	86.7% correct, 13.3% incorrect
Actual yes	90.7% correct, 9.3% incorrect (2.1% wrong position, 7.1% predict as no header)

Our algorithm correctly identified headers (or lack of which) in 89.5% of the tables. Among the 140 tables with a header, we correctly detected the header in 127 (90.7%) of them.

3. Entity column detection: Because the information in web tables is so diverse, random sampling from web tables may not have a good coverage in both structures and domains. Therefore in this part of the experiment, we create test set using tables extracted from Wikipedia site only. These tables contain structural multiple-domain data in high density. We only focus on precision here as recall is hard to evaluate. We randomly selected 200 tables which our system claims to have an entity column, while ensuring that no two tables are from the same Wikipedia page. Our judge evaluated the correctness of this claim for these 200 tables. Results showed that 11 tables actually do not have an entity column, and most of them have two or more main entities, which violate our assumption that there is only one entity column in these tables. Among the remaining 189 tables that do have an entity column, our algorithm correctly identifies the entity column in 165 tables(87.3%).

6.2 Search Engine

As we discussed before, we support four semantic components in a query: Concept, Entity, Attribute and Keyword. Different combinations of them lead to different types of query. In this experiment, we focus on one representative pattern of them: Concept + Attribute, because this pattern includes two main factors: the relation between entity and concept, and the relation between attribute and concept. Some examples of pattern include "politicians birthday", "companies industry" and "video games developer".

To avoid indexing all the tables on the web, we implemented the search engine using only Wikipedia's tables. For the 30 concepts, we select three attributes from each of them and generate corresponding queries. Considering we need to have a certain number of results for each query to make the evaluation possible, we ask users to first think of attributes for each concept that they want to query, and then choose at most three of them with enough number of results.

To evaluate both the precision and quality of ranking, we ask the users to give a score to each of the top 10 results of each query. Let $score_k$ denote the score for the result at position k, then the overall quality of a query result is defined as:

$$\frac{\sum_{k\in[1,n]} score_k/k}{\sum_{k\in[1,n]} 1/k}, \tag{12}$$

where n is the number of result.

In total 82 out of 90 queries return enough results. Among them, 53 queries return all-correct results, and 6 queries have a score lower than 0.6. The average score of all queries is 0.91. Figure 4 shows the average scores of queries for each concept. [5] Among all the 29 concepts which have result, 25 of the concepts received an average score higher than 0.8.

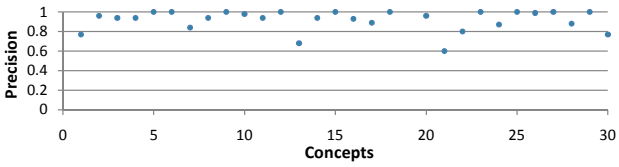

Figure 4: Average score of queries for each concept

To compare with an existing search engine, we submitted the same queries to Google. For each query, we manually judge the top

[5] some concepts don't have dots or bars in a chart because there aren't enough results for them (similarly hereinafter)

10 pages returned by Google to see if they contain the desired information. Since the query pattern, Concept + Attribute, is expecting a set of entities and their values, we are looking for pages containing forms, lists or categories about the queries class and attribute. It turned out that out of the 820 results (10 for each query), there are 32 results from Wikipedia and 81 results from non-wiki websites that we think contain the information we are expecting. Most of the pages from Wikipedia are Wikipedia's categories, which index entities by their values of one attribute. Many pages from non-wiki websites are also in the similar formats. To get the information we want in such pages, we need to first determine the value, then use it to access entities. This might be impractical in some cases, if we are not familiar with the concept, or we want to browse entities having various of values. This experiment also showed that traditional keyword search is inadequate in handling such semantic queries.

To better evaluate the quality of entities and values, we compare our results with Google Squared, which also outputs tables as results. Our prototype search engine merges statements with the same schema together as one table. We select at most top 10 schemata ranked by the highest score of their associated statements. For each query, we obtain a list of entities and their attribute values, and compare this list with the list returned by Google Squared. We manually checked the correctness of both lists and ensure that the entities and values in both lists are correct. Figure 5 shows the percentage of our result lists that is included by, overlapped with or disjoint with the list from Google Squared for all 30 concepts. We argue that our system not only correctly contains attribute-value information that Google Squared currently has for those 30 concepts, but also includes many additional attributes and values that are not found in Google Squared.

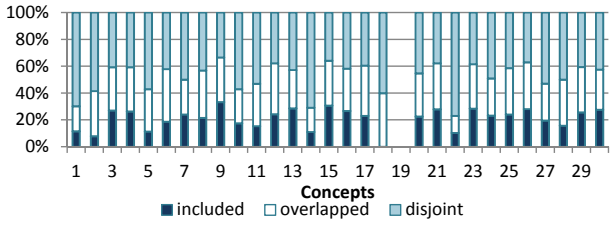

Figure 5: Comparing with Google Squared

6.3 Taxonomy Expansion

In this experiment, we evaluate how much our prototype system contributes in expanding the Probase taxonomy in terms of the number of entities and the number of attributes.

1. Entity expansion: To ensure a high quality, we select as seeds top 1000 entities ranked by ambiguity score $a_c(e)$ for each concept c, then use the plausibility score $p_c(e)$ to infer. Here we set γ_t to 0.6.

We run our algorithm for one iteration, and discovered 3.4 million entities which are already in Probase, but more importantly, 4.6 million new entities for nearly 20,000 Probase concepts. Among the 30 selected concepts, about 11,000 new entities were discovered from 28 of them (See Table 4), in which 1,410 (12.8%) are also found in Freebase by exact matching. Similar as before, we judge the top 20 entities (if available) manually for each concept, which are sorted by the maximal value of p_c, and then use the average score of them to indicate a concept's quality.

The result is shown in Fig 6. The average quality is 0.96, and all concepts have score higher than 0.875. Furthermore, new entities in 12 of the concepts are all correct. In essence, most of the newly discovered entities are correct.

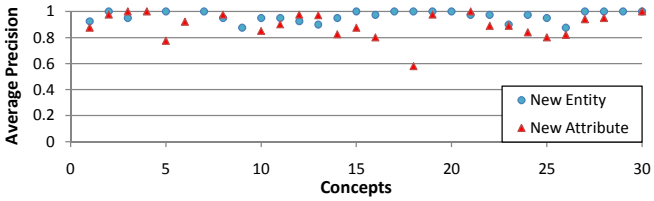

Figure 6: Precision of new entities and new attributes

2. Attribute expansion:

We again run our algorithm for one iteration, and discovered 0.15 million new attributes for nearly 14,000 concepts. About 2,700 new attributes were discovered for all the 30 concepts(See Table 4). Like before, we care more about the quality of top-ranked attributes than the general quality. For each concept, new attributes are first filtered though frequency and average $w_c(a)$, and then sorted according to the maximal $w_c(a)$. We still judge the top 20 attributes manually and use the average score to evaluate the concept. Here we set $\gamma_a = 0.4$.

The result is shown in Fig 6. We found new attributes for all 25 concepts, and the average quality of them is 0.90. The only outlier is the concept "guitarists", whose quality score is 0.58. That's because "guitarists" are often associated with "songs" in tables. But since Probase's coverage of songs is not good, guitarists are selected as the entity column instead which is not the best choice.

3. Entity relationship and similarity: We only present the result of relationship similarity here because the quality of entity pairs is included. We discovered about 1 million distinct relations, among which 42,000 of them have a size larger than 100 (with more than 100 entity pairs). We divide all the relationships into four levels by their size, then selected five relationships with clear semantic meanings from each level. For each relationship, we manually judge its top 10 synonyms which are sorted according to the similarity. The result is shown in Table 5.

Size	Avg top 5	Avg top 10
> 100,000	0.86	0.75
10,000 - 100,000	0.82	0.75
1000 - 10000	0.98	0.95
100 - 1000	0.9	0.66

Table 5: Relation Similarity

7. RELATED WORK

There has been some early work on information extraction from tables on the Web. Wang *et al.* [20] detects tables on the web by using machine learning methods. It uses features of layout, content type and word usage for training and testing different classifiers. However, the corpus only consists of about 10,000 tables collected from 200 web sites. Yoshida *et al.* [21] studied the problem of integrating tables of similar content into a big table. It uses EM method to decide the locations of attributes and values in a table by labeling the table format as a predefined structure, then group tables into several clusters according to their attributes and values. Similar attributes in a cluster are merged together to get a final big table. Another early work [10] focused on mining web tables in a specific domain: travel. It defines similarity between cells, rows, and columns, then uses the similarity to identify the schema of the table.

More recently, Cafarella *et al.* [8, 9] did some pioneering work in exploring tables on the Web. The goal is to decide if a table contains *relational* data. To achieve this, they tried to identify typed

columns in the table, which resulted in schemata for tables that are considered relational. They also studied an interesting application: searching over a corpus of tables. This work relied on correlation statistics inside the tables instead of external knowledge. Furthermore, each table is treated as a single entity (tables are atom units in query results) whereas we consider table as a set of entities and values. Syed *et al.* [19] used *Wikitology*, an ontology which combines some existing manually built knowledge systems such as DBPedia [5] and Freebase [7], to link cells in a table to Wikipedia entities. To resolve the ambiguity issue (a cell may match many Wikipedia entities), it finds the class or the domain of a column (or a row) through majority voting to achieve more accurate matching. Limaye *et al.* [14] used YAGO to annotate components of web tables as existing content in the taxonomy by using machine learning approaches. Specifically, they are interested in labeling cell text, column type, and relation between columns, then use the information to assist queries about relation. The cells and columns are treated equally, and the annotation results are constrained by the YAGO taxonomy. However, as we described in Section 1, people need knowledge, including common sense and worldly facts, to understand a table, especially when the table is taken out of its context. Without such knowledge, it is often impossible to truly understand the table, or to be able to perform the type of semantic search as described in this paper.

The work presented in this paper focuses on *understanding* tables. To obtain knowledge required for this task, we build it upon an extension of Probase [2], a rich ontology automatically derived from the Web to produce a concept map for the human mind. In our extension, we find attributes for each concept (there are more than 2 million of them in Probase). Some recent work has focused on finding attributes for given classes and concepts. In particular, Pasca *et al.* [15] finds class attributes starting from a few *seed* (entity, attribute) pairs for each class. Furthermore, a ranking mechanism [6] has been developed to score attributes derived from seeds.

Besides tables, some work has also been done to explore other sources of structured data on the Web, for example lists [11]. In fact, the work transforms lists to tables because table is a more general type of structured data. There is also research that transforms text to tables [17, 16]. In particular, Pyreddy *et al.* [17] uses heuristic rules to tag table components, and Pinto *et al.* [16] uses conditional random fields to locate boundaries, headers and rows of text table.

8. CONCLUSION

This paper presents a framework that attempts to harvest useful knowledge from the rich corpus of relational data on the Web: HTML tables. Through a multi-phase algorithm, and with the help of a universal probabilistic taxonomy called Probase, the framework is capable to understanding the entitles, attributes and values in many tables on the Web. With this knowledge, we built two interesting applications: a semantic table search engine which returns relevant tables from keyword queries, and a tool to further expand and enrich Probase. Our experiments indicate generally high performance in both table search results and taxonomy expansion. This showed that the proposed framework practically benefits knowledge discovery and semantic search.

9. REFERENCES

[1] `http://202.120.38.146/probase/tablesearch/Overview.html`.

[2] Omitted for double-blind review.

[3] Omitted for double-blind review.

[4] Wikipedia. `http://www.wikipedia.org`.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[6] K. Bellare, P. P. Talukdar, G. Kumaran, F. Pereira, M. Liberman, A. McCallum, and M. Dredze. Lightly-supervised attribute extraction. In *NIPS*, 2007.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.

[8] Michael J. Cafarella, Eugene Wu, Alon Halevy, Yang Zhang, and Daisy Zhe Wang. Uncovering the relational web. In *WebDB*, 2008.

[9] Michael J. Cafarella, Eugene Wu, Alon Halevy, Yang Zhang, and Daisy Zhe Wang. Webtables: Exploring the power of tables on the web. In *VLDB*, 2008.

[10] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale html texts. In *ICCL*, 2000.

[11] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. In *VLDB*, 2009.

[12] Oren Etzioni, Michael Cafarella, and Doug Downey. Webscale information extraction in knowitall (preliminary results). In *WWW*, 2004.

[13] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, pages 539–545, 1992.

[14] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. In *VLDB*, 2010.

[15] Marius Pasca. Organizing and searching the world wide web of facts - step two: Harnessing the wisdom of the crowds. In *WWW*, 2007.

[16] D. Pinto, A. McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *SIGIR*, 2003.

[17] Pallavi Pyreddy and W. Bruce Croft. Tintin: A system for retrieval in text tables. In *ICDL*, 1997.

[18] P. Singh, T. Lin, E. Mueller, G. Lim, T. Perkins, and W. Li Zhu. Open Mind Common Sense: Knowledge acquisition from the general public. *On the Move to Meaningful Internet Systems: CoopIS, DOA, and ODBASE*, pages 1223–1237, 2002.

[19] Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. Exploiting a web of semantic data for interpreting tables. In *Proceedings of the Second Web Science Conference*, 2010.

[20] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *WWW*, 2002.

[21] Minoru Yoshida, Kentaro Torisawa, and Junąfichi Tsujii. A method to integrate tables of the world wide web. In *International Workshop on Web Document Analysis*, 2001.