Recovering Semantics of Tables on the Web

Petros Venetis
Stanford University
venetis@cs.stanford.edu

Alon Halevy
Google Inc.
halevy@google.com

Jayant Madhavan
Google Inc.
jayant@google.com

Marius Paşca
Google Inc.
mars@google.com

Warren Shen
Google Inc.
whshen@google.com

Fei Wu
Google Inc.
wufei@google.com

Gengxin Miao
UC Santa Barbara
miao@umail.ucsb.edu

Chung Wu
Google Inc.
chungwu@gmail.com

ABSTRACT

The Web offers a corpus of over 100 million tables [6], but the meaning of each table is rarely explicit from the table itself. Header rows exist in few cases and even when they do, the attribute names are typically useless. We describe a system that attempts to recover the semantics of tables by enriching the table with additional annotations. Our annotations facilitate operations such as searching for tables and finding related tables.

To recover semantics of tables, we leverage a database of class labels and relationships automatically extracted from the Web. The database of classes and relationships has very wide coverage, but is also noisy. We attach a class label to a column if a sufficient number of the values in the column are identified with that label in the database of class labels, and analogously for binary relationships. We describe a formal model for reasoning about when we have seen sufficient evidence for a label, and show that it performs substantially better than a simple majority scheme. We describe a set of experiments that illustrate the utility of the recovered semantics for table search and show that it performs substantially better than previous approaches. In addition, we characterize what fraction of tables on the Web can be annotated using our approach.

1. INTRODUCTION

The Web offers a corpus of over 100 million high-quality tables on a wide variety of topics [6]. However, these tables are embedded in HTML and therefore their meaning is only described in the text surrounding them. Header rows exist in few cases, and even when they do, the attribute names are typically useless.

Without knowing the semantics of the tables, it is very difficult to leverage their content, either in isolation or in combination with others. The challenge initially arises in *table search* (for queries such as *countries population*, or *dog breeds life span*), which is the first step in exploring a large collection of tables. Search engines typically treat tables like any other text fragment, but signals that work well for text do not apply as well to table corpora. In particular, document search often considers the proximity of search terms on the page to be an important signal, but in tables the column headers apply to every row in the table even if they are textually far away. Furthermore, unlike text documents, where small changes in the document structure or wording do not correspond to vastly different content, variations in table layout or terminology change the semantics significantly. In addition to table search, knowing the semantics of tables is also necessary for higher-level operations such as combining tables via join or union.

In principle, we would like to associate semantics with each table in the corpus, and use the semantics to guide retrieval, ranking and table combination. However, given the scale, breadth and heterogeneity of the tables on the Web, we cannot rely on hand-coded domain knowledge. Instead, this paper describes techniques for automatically recovering the semantics of tables on the Web. Specifically, we add annotations to a table describing the sets of entities the table is modeling, and the binary relationships represented by columns in the tables. For example, in the table of Figure 1 we would add the annotations `tree species`, `tree`, and `plant` to the first column and the annotation `is known as` to describe the binary relation represented by the table.

Hazel Alder	*Alnus serrulata*
Green Ash	*Fraxinus pennsylvanica*
White Ash	*Fraxinus americana*
Baldcypress	*Taxodium distichum*
Beech	*Fagus grandifolia*
River Birch	*Betula nigra*
Boxelder	*Acer negundo*
Red Cedar	*Juniperus virginiana*
Black Cherry	*Prunus serotina*
Sweet Cherry	*Prunus avium*
Crab Apple	*Malus coronaria*
Crapemyrtle	*Lagerstroemia indica*
Dogwood	*Cornus florida*

Figure 1: An example table on the Web, associating the common names of trees to their scientific names. Full table at `http://www.hcforest.sailorsite.net/Elkhorn.html`.

The key insight underlying our approach is that we can use facts extracted from text on the Web to interpret the tables. Specifically, we leverage two databases that are extracted from the Web: (1) an isA database that contains a set of pairs of the form (instance, class), and (2) a relations database of triples of the form (argument1, predicate, argument2). Because they are extracted from the Web, both of these databases have very broad coverage of topics and instances, but are also noisy. We use them to annotate columns as follows. We label a column A with class C in the isA database if a substantial fraction of the cells in a column A are labeled with a class C in the isA database. We label the relationship between columns A and B with R if a substantial number of pairs of values from A and B occur in extractions of the form (a, R, b) in the relations database. We describe a formal model that lets us determine how much evidence we need to find in the extracted databases in order

Proceedings of the VLDB Endowment, Vol. 4, No. 9

to deem a label appropriate for a column or a pair of columns. In particular, the model addresses the challenge that the extracted databases are not a precise description of the real world or even of the Web, since some entities occur more frequently on the Web than others.

We show experimentally that the labels we generate describe the contents of the table well and are rarely explicit in the table itself. We show that the labels are even more accurate when we consider only the ones that are associated with a column in the table that is the *subject* of the table. Based on this, we build a table search engine with much higher precision than previous approaches. In summary, we make the following contributions:

- We propose an approach that partially recovers semantics of structured data on the Web in order to improve table search and enable more advanced operations. Our approach uses information extracted from text on the Web.
- We describe a formal model for reasoning about the evidence underlying our annotations.
- We describe a table-search algorithm based on the recovered semantics.
- We describe an experimental evaluation on a corpus of 12.3 million tables extracted from the Web. We show: (1) that we obtain meaningful labels for tables that rarely exist in the tables themselves, (2) a characterization of the portion of tables on the Web that can be annotated using our method, and that it is an order of magnitude larger than is possible by mapping tables to Wikipedia classes or Freebase attribute columns, and (3) that considering the recovered semantics leads to high precision search with little loss of recall of tables in comparison to document based approaches.

Roadmap: Section 2 defines our terminology and the table search problem. Section 3 describes how we recover semantics of tables and our formal model. Section 4 describes our experimental evaluation. Section 5 describes related work, and Section 6 concludes.

2. PROBLEM SETTING

We begin by describing the Web table corpus and the problems of annotating tables and table search.

Table corpus: Each table in our corpus is a set of rows, and each row is a sequence of cells with data values (see Figure 1 for an example). Tables may be semi-structured and have very little metadata. Specifically:

- We do not have a name for the table (i.e., the relationship or entities that it is representing).
- There may not be names for attributes, and we may not even know whether the first row(s) of the table are attribute names or data values (as in Figure 1).
- The values in a particular row of a column will typically be of a single data type, but there may be exceptions. Values may be taken from different domains and different data types. In addition, we often also see sub-header rows of the type one would see in a spreadsheet (see Figure 5 in the appendix).
- The quality of tables in the corpus varies significantly, and it is hard to determine whether HTML table tags are used for high-quality tabular content or as a formatting convenience.

Annotating tables: Our goal is to add annotations to tables to expose their semantics more explicitly. Specifically, we add two kinds of annotations. The first, *column labels* are annotations that represent the set of entities in a particular column. For example, in the table of Figure 1 possible column labels are `tree`, `tree species` and `plant`. The second, *relationship labels* represent the binary relationship that is expressed by a pair of columns in the table. For example, a possible relationship label in the table of Figure 1 is `is known as`. We note that our goal is to produce *any* relevant label that appears on the Web, and therefore be able to match more keyword queries that users might pose. In contrast, previous work [17] focused on finding a *single* label from an ontology (YAGO [26]).

Typically, tables on the Web have a column that is the subject of the table. The subject column contains the set of entities the table is about, and the other columns represent binary relationships or properties of those entities. We have observed that over 75% of the tables in our corpus exhibit this structure. Furthermore, the subject column need not be a key — it may contain duplicate values.

Identifying a subject column is important in our context because the column label we associate with it offers an accurate description of what the table is about, and the binary relationships between the subject column and other columns reflect the properties that the table is representing. Hence, while our techniques do not require the presence of a subject column, we show that the accuracy of our annotations and resulting table search are higher when the subject column is identified.

Table search: We test the quality of our annotations by their ability to improve table search, the most important application that the annotations enable. We assume that queries to table search can be posed using any keyword because it is unreasonable to expect users to know the schemata of such a large collection of heterogeneous tables in such a vast array of topics.

In this work we consider returning a ranked list of tables in response to a table search query. However, the ability to retrieve tables based on their semantics lays the foundation for more sophisticated query answering. In particular, we may want to answer queries that require combining data from multiple tables through join or union. For example, consider a query asking for the relationship between the incidence of malaria and the availability of fresh water. There may be a table on the Web for describing the incidence of malaria, and another for access to fresh water, but the relationship can only be gleaned by joining two tables.

We analyzed Google's query stream and found that queries that could be answered by table search fall into two main categories: (1) find a property of a set of instances or entities (e.g., *wheat production of African countries*), and (2) find a property of an individual instance (e.g., *birth date of Albert Einstein*). This paper focuses on queries of the first kind, and we assume they are of the form (C, P), where C is a string denoting a class of instances and P denotes some property associated with these instances. Both C and P can be *any* string rather than being drawn from a particular ontology, but we do not consider the problem of transforming an arbitrary keyword query into a pair (C, P) in this paper. We also note that there are millions of queries of both kinds being posed every day.

While our techniques can also be used to help answering the second kind of queries, there are many other techniques that come into play [1, 12]. In particular, answers to queries about an instance and a property can often be extracted from free text (and corroborated against multiple occurrences on the Web).

Finally, we note that we do not consider the problem of blending results of table search with other Web results.

3. ANNOTATING TABLES

At the size and breadth of the table corpus we are considering, manually annotating the semantics of tables will not scale. The key idea underlying our approach is to automatically annotate tables by leveraging resources that are already on the Web, and hence

have similar breadth to the table corpus. In particular, we use two different data resources: (1) an isA database consisting of pairs (instance, class) that is extracted by examining specific linguistic patterns on the Web, and (2) a relations database consisting of triplets of the form (argument1, predicate, argument2) extracted without supervision from the Web. In both databases, each extraction has a score associated with it describing the confidence in the extraction. The isA database is used to produce column labels and the relations database is used to annotate relationships expressed by pairs of columns. Importantly, our goal is not necessarily to recover a single most precise semantic description (i.e., we cannot compete with manual labeling), but just enough to provide useful signals for search and other higher-level operations.

Sections 3.1 and 3.2 describe how the isA database and relations database are created, respectively. In Section 3.3 we consider the problem of how evidence from the extracted databases should be used to choose labels for tables. Since the Web is not a precise model of the real world and the algorithms used to extract the databases from the Web are not perfect, the model shows which evidence should be weighed more heavily than others in determining possible labels. As we described earlier, when labels are associated with a subject column, they are even more indicative of the table's semantics. We describe an algorithm for discovering subject columns in Appendix C.

3.1 The isA database

The goal of column labels is to describe the class of entities that appear in that column. In Figure 1 the labels `tree`, `tree species` and `plant` describe the entities in the first column and may correspond to terms used in searches that should retrieve this table. Recall that the isA database is a set of pairs of the form (instance, class). We refer to the second part of the pair as a *class label*. We assign column labels to tables from the class labels in the isA database. Intuitively, if the pairs (I, C) occur in the isA database for a substantial number of values in a column A, then we attach C as a column label to A. We now describe how the isA database is constructed.

We begin with techniques such as [21] to create the isA database. We extract pairs from the Web by mining for patterns of the form:

$$\langle [..]\ C\ [\text{such as}|\text{including}]\ I\ [\text{and}|,|.]\rangle,$$

where I is a potential instance and C is a potential class label for the instance (e.g., cities such as Berlin, Paris and London).

To apply such patterns, special care needs to be paid to determining the boundaries of C and of I. Boundaries of potential class labels, C, in the text are approximated from the part-of-speech tags (obtained using the TnT tagger [4]) of the sentence words. We consider noun phrases whose last component is a plural-form noun and that are not contained in and do not contain another noun phrase. For example, the class label `michigan counties` is identified in the sentence [..] michigan counties such as van buren, cass and kalamazoo [..]. The boundaries of instances, I, are identified by checking that I occurs as an entire query in query logs. Since users type many queries in lower case, the collected data is converted to lower case. These types of rules have also been widely used in the literature on extracting conceptual hierarchies from text [13, 25].

To construct the isA database, we applied patterns to 100 million documents in English using 50 million anonymized queries. The extractor found around 60,000 classes that were associated with 10 or more instances. The class labels often cover closely-related concepts within various domains. For example, `asian countries`, `east asian countries`, `south asian countries`, and `southeast asian countries` are all present in the extracted data. Thus, the extracted class labels correspond to both a broad and relatively deep conceptualization of the potential classes of interest to Web search users, on one hand, and human creators of Web tables, on the other hand. The reported accuracy for class labels in [21] is >90% and the accuracy for class instances is almost 80%.

To improve the coverage of the database beyond the techniques described in [21], we use the extracted instances of a particular class as seeds for expansion by considering additional matches in Web documents. We look for other patterns on the Web that match more than one instance of a particular class, effectively inducing document-specific extraction wrappers [16]. For example, we may find the pattern $\langle$headquartered in $I\rangle$ and be able to mine more instances I of the class label `cities`. The candidate instances are scored across all documents, and added to the list of instances extracted for the class label [29]. This increases coverage with respect to instances, although not with respect to class labels.

Given the candidate matches, we now compute a score for every pair (I, C) using the following formula [20]:

$$Score(I, C) = Size(\{Pattern(I, C)\})^2 \times Freq(I, C). \quad (1)$$

In the formula, $Pattern(I, C)$ is the set of different patterns in which (I, C) was found. $Freq(I, C)$ is the frequency count of the pair, but since high frequency counts are often indicative of near-duplicate sentences appearing on many pages, we compute it as follows. We compute a sentence fingerprint for each source sentence, by applying a hash function to at most 250 characters from the sentence. Occurrences of (I, C) with the same sentence fingerprint are only counted once in $Freq(I, C)$.

3.2 The relations database

We also want to annotate tables with the set of relationships that it represents between pairs of entities. For example, the table in Figure 1 represents the relationship `is known as` between trees and their scientific names. In general, two types of relationships are common in tables on the Web: symbolic (e.g., `capital of`) and numeric (e.g., `population`). In what follows we use the relations database to obtain labels for the symbolic relationships (we comment on numeric relationships in Section 6).

Intuitively, given two columns, A and B, we look at corresponding pairs of values in the columns. If we find that the relation (a, R, b) is extracted for many rows of the table, then R is a likely label for the relationship represented by A and B.

We use Open Information Extraction (OIE) to extract triples for the relations database. Unlike traditional information extraction that outputs instances of a *given* relation, OIE extracts any relation using a set of relations-independent heuristics. In our implementation, we use the TextRunner open extraction system [2]. As reported in [2], TextRunner has precision around 73.9% and recall around 58.4%. In Appendix B we provide additional details about TextRunner.

3.3 Evaluating candidate annotations

The databases described above provide evidence from the Web that a particular label applies to a column, or that a particular binary relationship is represented by a pair of columns. However, the immediate question that arises is how much evidence is enough in order to assign a label to a column or pair of columns, or alternatively, how to rank the candidate labels.

If the isA and relations databases were a precise description of the real world, then we would require a label to apply to *all* rows of a table before it is assigned. However, the databases have two kinds of imprecision: first, the Web is *not* an accurate description of the world, and second, no matter how good the extractors are, they will miss some facts that are mentioned on the Web. Consider the effect

of the first kind of imprecision. Paris and France are mentioned very frequently on the Web and therefore we expect to find sentences on the Web that say that Paris is the capital of France. However, Lilongwe and Malawi are not mentioned as often, and therefore there is a smaller likelihood of finding sentences that say that Lilongwe is the capital of Malawi. Hence, if we have a table that includes a row for Paris, France and one for Lilongwe, Malawi, but we do not find (Lilongwe, capital of, Malawi) in the relations database, that should not be taken as strong evidence against assigning the label `capital of` to that pair of columns.

The second kind of imprecision stems from the fact that ultimately, the extractors are based on some kind of rules that may not extract everything that is said on the Web. For example, to extract cities, we look for patterns of the form ⟨cities such as I⟩, which may not be found for rare entities such as Lilongwe. In addition, some entities are simply not mentioned in such patterns at all. For example, there are many tables on the Web describing the meaning of common acronyms, but there are very few sentences of the form ⟨acronyms such as I⟩.

The model we describe below lets us reason about how to interpret the different kind of positive and negative evidence we find in our extracted database. We use a maximum-likelihood model based on the following intuition. A person constructing a table in a Web page has a particular intent ("schema") in mind. The intent is to describe properties of instances of an entity class. The maximum-likelihood model attempts to assign class labels to a column given the contents the person has used to populate the column. The best label is therefore the one that, if chosen as part of the underlying intent, is most likely to have resulted in the observed values in the column. Hence, we are trying to infer the intent of the table designer based on the evidence they have given us.

We begin by considering the problem of assigning class labels to a column. Assigning labels to binary relationships is analogous and covered in Appendix B. Let $V = \{v_1, \ldots, v_n\}$ be the set of values in a column A. Let $l_1, \ldots, l_m$ be all possible class labels.

To find the best class label, we use the maximum likelihood hypothesis [19], i.e., the best class label $l(A)$ is one that maximizes the probability of the values given the class label for the column:

$$l(A) = \arg\max_{l_i} \{\Pr[v_1, \ldots, v_n \mid l_i]\}.$$

We assume that each row in the table is generated independently given the class label for the column, (and thus $\Pr[v_1, \ldots, v_n \mid l_i] = \prod_j \Pr[v_j \mid l_i]$). This is, again, a reasonable assumption in our context because tables that are relational in nature are likely to have dependencies between column values in the same row, rather than across rows. Further, from Bayes rule, we know that $\Pr[v_j \mid l_i] = \frac{\Pr[l_i \mid v_j] \times \Pr[v_j]}{\Pr[l_i]}$. It follows that:

$$\Pr[v_1, \ldots, v_n \mid l_i] = \prod_j \frac{\Pr[l_i \mid v_j] \times \Pr[v_j]}{\Pr[l_i]} \propto \prod_j \frac{\Pr[l_i \mid v_j]}{\Pr[l_i]}.$$

The product term $\prod_j \Pr[v_j]$ applies identically to each of the labels. Hence, it follows that $l(A) = \arg\max_{l_i} \prod_j \frac{\Pr[l_i \mid v_j]}{\Pr[l_i]}$.

We assign a score $U(l_i, V)$ to each class that is proportional to the expression in the above equation. We normalize them to sum up to 1, i.e.,

$$U(l_i, V) = K_s \prod_j \frac{\Pr[l_i \mid v_j]}{\Pr[l_i]}, \tag{2}$$

where normalization constant K_s is such that $\sum_i U(l_i, V) = 1$.

The probability $\Pr[l_i]$ can be estimated from the scores in the isA database (see Equation 1). However, estimating the conditional probability $\Pr[l_i \mid v_j]$ is more challenging. While a simple estimator is $\frac{Score(v_j, l_i)}{\sum_k Score(v_j, l_k)}$, it has two problems. First, when computing the maximum likelihood hypothesis, since we are multiplying each of the $\Pr[l_i \mid v_j]$, we cannot have any of the probabilities be 0. Second, since information extracted from the Web is inherently incomplete, it is likely that there are values for which there is an incomplete set of labels in the isA database.

To account for the incompleteness, we *smooth* the estimates for conditional probabilities:

$$\Pr[l_i \mid v_j] = \frac{K_p \times \Pr[l_i] + Score(v_j, l_i)}{K_p + \sum_k Score(v_j, l_k)},$$

where K_p is a smoothing parameter.

The above formula ensures that in the absence of any isA extractions for v_j, the probability distribution of labels tends to be the same as the prior $\Pr[l_i]$. As a result, values with no known labels are not taken as negative evidence and do not contribute to changing the ordering among best hypotheses. On the other hand, if there are many known class-label extractions for v_j, its conditional probabilities tend towards their true value and hence such v_j contribute significantly (positively or negatively) in selecting the best class labels. As the score in the isA database increases (with increased extractions from the Web), the conditional probability estimator depends more on the scores. The parameter K_p controls how sensitive the probabilities are to low extraction scores. If we are to assume that extractions from the Web are mostly true (but incomplete), then we can set K_p to be very low (say 0.01).

Finally, we need to account for the fact that certain expressions are inherently more popular on the Web and can skew the scores in the isA database. For example, for a value v with two labels with $Score(v, l_1) = 100$ and $Score(v, l_2) = 10{,}000$, a simple fraction with result in $\Pr[l_1 \mid v] \ll \Pr[l_2 \mid v]$. We refine our estimator further to instead use the logarithm of the scores, i.e.,

$$\Pr[l_i \mid v_j] = \frac{K_p \times \Pr[l_i] + \ln(Score(v_j, l_i) + 1)}{K_p + \sum_k \ln(Score(v_j, l_k) + 1)}. \tag{3}$$

The $+1$ in the logarithm prevents $\ln 0$. As before, the probabilities can be normalized to sum to 1.

To determine the prior probabilities of class labels $\Pr[l_i]$, we add the scores across all values for that label, i.e.,

$$\Pr[l_i] \propto \sum_j \ln(Score(v_j, l_i) + 1 + \delta). \tag{4}$$

We use $1 + \delta$ to ensure that $\Pr[l_i] \neq 0$. The probabilities are normalized such that $\sum_i \Pr[l_i] = 1$.

Given the set of values in a column, we estimate the likelihood score U for each possible label (Equation 2). We consider only the labels that have a normalized likelihood score greater than a threshold t_l and rank the labels in decreasing order of their scores.

4. EXPERIMENTS

We evaluate the quality of the table annotations (Section 4.1) and their impact on table search (Section 4.2).

Table corpus: Following [6], we constructed a corpus of HTML tables extracted from a subset of the crawl of the Web. We considered pages in English with high page rank. From these, we extracted tables that were clearly not HTML layout tables, and then filtered out empty tables, form tables, calendar tables, tiny tables (with only 1 column or with less than 5 rows). We were left with about 12.3 million tables. We estimate that this corpus represents about a tenth of the high-quality tables on the Web as of late 2010.

4.1 Column and relation labels

We discuss the quality of the labels assigned with the isA and relations databases in Section 4.1.1. In Section 4.1.2 we show that our method labels an order of magnitude more tables than is possible with Wikipedia labels and many more tables than Freebase. In Section 4.1.3 we show that the vast majority of the remaining tables can either be labeled using a few domain specific rules or do not contain useful content.

4.1.1 Label quality

We compare three methods for assigning labels. The first, denoted Model, is the likelihood model described in Section 3.3. The second, denoted Majority, requires that at least $t\%$ of the cells of the column have a particular label. Of these, the algorithm ranks the labels according to a $MergedScore(C) = \sum_L \frac{1}{Rank(C,L)}$ (if C has not been assigned to cell content L, then $Rank(C, L) = \infty$). After experimenting with different values, we observed that the Majority algorithm performs best when $t = 50$. We also examined a Hybrid method that uses the ranked list of the Majority method concatenated with the ranked list of the Model (after removing labels outputted by Majority). As we will explain in the following, the Hybrid method performs better than both the Model and the Majority methods.

Gold standard: To create a gold standard, we considered a random sample of tables and removed those that did not have any class or relations labels assigned by run R_{10}, the Majority algorithm with $t = 10\%$ (i.e., a very permissive labeling — for examples see Table 5 in Appendix C.2). We then manually removed tables whose subject columns have been incorrectly identified or do not correspond to any meaningful concept. For each of the remaining 168 tables, we presented to human annotators the result of R_{10}. The annotators mark each label as *vital*, *okay*, or *incorrect*. For example, given a table column containing the cells {Allegan, Barry, Berrien, Calhoun, Cass, Eaton, Kalamazoo, Kent, Muskegon, Saint Joseph, Van Buren}, the assigned class labels `southwest michigan counties` and `michigan counties` are marked as *vital*; labels `counties` and `communities` as *okay*; and `illinois counties` and `michigan cities` as *incorrect*. In addition, the annotators can manually enter any additional labels that apply to the table column, but are missing from those returned by any of the experimental runs. The resulting gold standard associates the 168 tables with an average of 2.6 *vital* and 3.6 *okay* class labels, with 1.3 added manually by the annotators. For the relations labels we had an average of 2.1 *vital* and 1.4 *okay* labels; since there were many options by our relations database, we did not add any new (manual) annotations to the binary relations labels.

Evaluation methodology: For a given table, the actual evaluation consists of automatically comparing the ranked lists of labels produced by an experimental run to the labels available in the gold standard. To compute precision, a retrieved label is assigned a score of 1, if it was marked as *vital* or manually added to the gold standard; 0.5, if it was marked as *okay*; or 0, otherwise [21]. Similarly, recall is computed by considering a label as relevant (score 1) if it was marked as *vital* or *okay* or was manually added to the gold standard, or irrelevant (score 0) otherwise.

Results: Figure 2 summarizes the performance results for the three algorithms. We varied the precision and recall by considering top-k labels for values of k between 1 and 10; k increases from the left of the graph to its right.

We observed that Majority (with $t = 50$) has relatively high precision but low recall (it labeled only 30% of the 168 tables). This is due to the requirement that a label must be given to 50% of the rows. In addition, Majority tends to output general labels (e.g., `compound chemical` vs. `antibiotic`) because they are more common on the Web and more rows are likely to agree on them. Still, its labels are generally of high quality. On the other hand, Model tends to do well in the cases where there are good labels, but they do not appear for a majority of rows in the table, in a sense, where more subtle reasoning is required. Consequently, Hybrid gives the best of both methods.

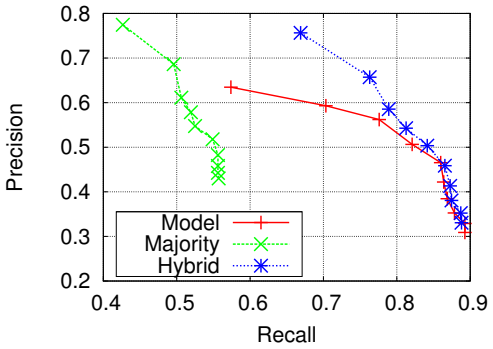

Figure 2: Precision/recall diagram for class labels for various algorithms and top-k values.

We obtained similar results for binary relationships (which we omit due to space considerations) except that Majority did not perform well. The reason is that our extractions are more sparse than in the unary case so it was harder to find labels that occur for 50% of the rows. Still, we obtained precision of 0.45 and recall of 0.7.

One may wonder if the class labels are not redundant with information that is already on the Web page of the table. In fact, there are only about 60,000 tables in our corpus (4%) where all class labels already appears in the table header, and only about 120,000 tables (8%) where a label appears anywhere in the body of the Web page. Hence, assigning class labels adds important new information.

4.1.2 Labels from ontologies

Next, we compare the coverage of our labeling to what can be obtained by using a manually created ontology. Currently, the state-of-the-art, precision-oriented isA database is YAGO [26], which is based on Wikipedia. Table 1 compares the labeling of tables using YAGO vs. the isA database extracted from the Web. Our Web-extracted isA database is able to assign labels to the subject columns of almost 1.5 million tables (out of 12.3 million tables we have at hand), while YAGO assigns labels to ∼185 thousand tables (an order of magnitude difference). This is explained by the very large coverage that our Web-extracted repository has in terms of instances (two orders of magnitude larger than YAGO).

	Web-extracted	YAGO	Freebase
Labeled subject columns	1,496,550	185,013	577,811
Instances in ontology	155,831,855	1,940,797	16,252,633

Table 1: Comparing our isA database and YAGO

For the binary relations, we were able to assign about 3 times as many labels for pairs of columns than Freebase (2.1M compared to 800K). We also examined the quality of the binary labels on our gold standard that included 128 binary relations involving a subject column. Our model detected 83 of them (64.8%) correctly (assigning *vital* or *average* binary labels), while Freebase only managed to detect 37 of them (28.9%) correctly.

We also compared our labeling on the same data sets (wiki manual and Web manual data sets) used in [17], where the authors proposed using YAGO to label columns in tables. These data sets have tables from Wikipedia and tables that are very related to Wikipedia tables, and hence we expect YAGO to do relatively well. Still, we achieved F1 measure of 0.67 (compared to the 0.56 reported in [17]) on the wiki manual data set, and 0.65 for the Web manual data set (compared to 0.43), both for the top-10 class labels returned by the majority-based algorithm. We note that using YAGO will result in higher precision. For the goal of table search though, coverage (and hence recall) is key.

4.1.3 The unlabeled tables

Our methods assigned class labels to approximately 1.5 million tables out of the 12.3 in our corpus when only subject columns are considered, and 4.3 million tables otherwise. We investigated why the other tables were not labeled, and most importantly, whether we are missing good tables in the unlabeled set. We discovered that the vast majority of these tables were either not useful for answering (C, P) queries, or can be labeled using a handful of domain-specific methods. Table 2 summarizes the main categories of the unlabeled tables.

Category	Sub-category	# tables (M)	% of corpus
Labeled	Subject column	1.5	12.20
	All columns	4.3	34.96
Vertical		1.6	13.01
Extractable	Scientific Publications	1.6	13.01
	Acronyms	0.043	0.35
Not useful		4	32.52

Table 2: Class label assignment to various categories of tables

First, we found that many of the unlabeled tables are *vertical* tables. These tables contain (attribute name, value) pairs in a long 2-column table (see Figure 3 in the appendix). We developed an algorithm for identifying such tables by considering tables that had at least two known attribute names in the left columns (the known attribute names were mined from Wikipedia and Freebase). This process identified around 1.6 million tables. After looking at a random sample of over 500 of these tables, we found that less than 1% of them would be useful for table-search queries.

Next, we found a few categories where the entities are too specific to be in the isA database. In particular, the most voluminous category is tables about publications or patents (1.45 million tables). It turns out that these can be identified using simple heuristics from very few sites. As another, much smaller category, we found 43,000 tables of acronyms on one site. Thus, extending our work to build a few domain-specific extractors for these tables could significantly increase the recall of our class assignment.

Among the remaining 4 million tables we found (based on a random sample of 1,000) that very few of them are useful for (C, P) queries. In particular, we found that many of these tables have enough text in them to be retrieved by traditional search techniques. Examples of such categories include course description tables (with the course number and university on the page) and comments on social networks, bug reports and job postings.

Hence, in summary, though we have annotated about a sixth of our tables, our results suggest that these are the *useful* content for table-search queries.

4.2 Table search

We now describe the impact of our annotations on table search. We built a table search engine which we refer to as TABLE, that leverages the annotations on tables. Given a query of the form (C, P), where C is a class name and P is a property, TABLE proceeds as follows:

Step 1: Consider tables in the corpus that have the class label C in the top-k class labels according to Section 3.3.[1] Note that tables that are labeled with C may also contain only a subset of C or a named subclass of C.

Step 2: We rank the tables found in step 1 based on a weighted sum of the following signals: occurrences of P on the tokens of the schema row, occurrences of P on the assigned binary relations of the table, page rank, incoming anchor text, number of rows and tokens found in the body of table and the surrounding text. The weights were determined by training on a set of examples. We note that in our current implementation we require that there be an occurrence of P in the schema row (which exist in 71% of the tables [7]) or in the assigned binary relations of the table.

In principle, it would also be possible to estimate the size of the class C (from our isA database) and to try to find a table in the result whose size is close to C. However, this heuristic has several disadvantages. First, the isA database may have only partial knowledge of the class, and therefore the size estimate may be off. Second, it is very common that the answer is not in a table that is precisely about C. For example, the answer to (*african countries, GDP*) is likely to be in a table that includes all the countries in the world, not only the African countries. Hence, we find that in general longer tables tend to provide better answers.

We compare TABLE with three other methods: (1) GOOG: the results returned by `www.google.com`, (2) GOOGR: the intersection of the table corpus with the top-1,000 results returned by GOOG, and (3) DOCUMENT: document-based approach proposed in [6]. The document-based approach considers several signals extracted from the document in the ranking, including hits on the first two columns, hits anywhere in the table (with a different weight) and hits on the header of the subject column.

Query set: To construct a realistic set of user queries of the form (C, P), we analyzed the query logs from Google Squared, a service in which users search for structured data. We compiled a list of 100 queries (i.e., class names) submitted by users to the website. For each class name, each of the authors identified potential relevant property names. We then randomly selected two properties for each class name to create a test set of 200 class-property queries. We chose a random subset of 100 out of the 200 queries (see Appendix D for a complete listing).

Evaluation methodology: We performed a user study to compare the results of each algorithm. For the purpose of this experiment, each algorithm returns Web pages (if an algorithm originally returned Web tables, we now modified it to return the Web pages containing those Web tables). For each of the 100 queries, we retrieved the top-5 results using each of TABLE, DOCUMENT, GOOG, and GOOGR. We combine and randomly shuffle these results, and present to the user this list of at most 20 search results (only GOOG is always guaranteed to return 5 results). For each result, the user had to rate whether it was *right on* (has all information about a large number of instances of the class and values for the property), *relevant* (has information about only some of the instances, or of properties that were closely related to the queried property), or *irrelevant*. In addition, the user marked if the result, when *right on* or

[1] As an extension, when C is not in the isA database, TABLE could search for other class names that are either the correct spelling of C or could be considered related — these extensions are currently not supported in TABLE.

Method	All Ratings				Ratings by Queries				Query Precision			Query Recall		
	Total	(a)	(b)	(c)	Some Result	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
TABLE	175	69	98	93	49	24	41	40	**0.49**	**0.84**	**0.82**	0.53	0.55	**0.63**
DOCUMENT	399	24	58	47	93	13	36	32	0.14	0.39	0.34	0.29	0.48	0.51
GOOG	493	63	116	52	100	32	52	35	0.32	0.51	0.35	**0.71**	**0.69**	0.56
GOOGR	156	43	67	59	65	17	32	29	0.26	0.49	0.45	0.38	0.43	0.46

Table 3: Results of user study: The columns under All Ratings present the number of results (totalled over the 3 users) that were rated to be (a) *right on*, (b) *right on* or *relevant*, and (c) *right on* or *relevant* and *in a table*. The Ratings by Queries columns aggregate ratings by queries: the sub-columns indicate the number of queries for which at least two users rated a result similarly (with (a), (b) and (c) as before). The Precision and Recall are as defined in Section 4.2.

relevant, was contained *in a table*. The results for each query were rated independently by three separate users.

Note that by presenting a combined, shuffled list of search results, and asking the user to rate the result Web documents, we can determine which algorithm produced each result. We cannot present the users directly with the extracted tables, because GOOG does not always retrieve results with tables. Further, we do not ask users to directly compare the ranked lists of results listed separately by approach, since it might be possible for a rater to work out which algorithm produced each list. Thus, we are able to achieve a fair comparison to determine which approach can retrieve information (not just tables) that is relevant to a user query.

Precision and recall: The results of our user evaluation are summarized in Table 3. We can compare the different methods using measures similar to the traditional notions of precision and recall. Suppose $N_q(m)$ was the number of queries for which the method m retrieved some result, $N_q^a(m)$ was the number of queries for which m retrieved some result that was rated *right on* by at least two users, and $N_q^a(*)$ is the number of queries for which some method retrieved a result that was rated *right on*. We define $P^a(m)$ and $R^a(m)$ to be:

$$P^a(m) = \frac{N_q^a(m)}{N_q(m)}, \qquad R^a(m) = \frac{N_q^a(m)}{N_q^a(*)}.$$

Note that we can likewise define $P^b(m)$ and $R^b(m)$ by considering results that were rated *right on* or *relevant* and $P^c(m)$ and $R^c(m)$ by considering results that were rated *in a table* (*right on* or *relevant*). Note that each $P(m)$ and $R(m)$ roughly correspond to traditional notions of precision and recall.

In our experiments, we found $N_q^a(*) = 45$ (*right on*), $N_q^b(*) = 75$ (*right on* or *relevant*), and $N_q^c(*) = 63$ (*in a table*). The resulting values for precision and recall are listed in Table 3. Note that we could likewise define these measures in terms of the number of results (and the patterns are similar).

Results: As demonstrated in Table 3, TABLE has the highest precision (0.84 when considering *right on* and *relevant* results). This result shows that even modest recovery of table semantics leads to very high precision. GOOG on the other hand has a much higher recall, but a lower precision.

We note that the recall performance of GOOG is based on retrieving Web pages that are relevant Web pages (not necessarily tables that are *right on*). In fact, the precision of GOOG is lower, if we only consider the *right on* ratings (0.32). If we only consider queries for which the relevant information was eventually found in a table, TABLE has both the highest precision (0.82) and highest recall (0.63) and clearly outperforms GOOG. This result shows that not only does TABLE have high precision, but it does not miss many tables that are in the corpus. Hence, we can use TABLE to build a search service for tables, and when it returns too few answers, we can fall back on general Web search.

Observe that DOCUMENT does not perform well in comparison to either TABLE or GOOG. This is likely because DOCUMENT (as described in [6]) was designed to perform well for instance queries. It does not have the benefit of class labels, which are no doubt important for class-property queries. It essentially boils down to be like GOOG, but with a far smaller corpus (only our ∼4.3 million extracted tables), and hence has poor performance.

GOOGR in general has a higher precision and lower recall than GOOG. GOOGR filters the results from GOOG to only include Web pages that have tables with class labels. Thus, it will retrieve information when present in table (higher precision, as they are excellent at answering class-property queries), but omit relevant Web pages without tables.

Our results clearly demonstrate that whenever there is a table that satisfies a class-property query, our table search algorithm is likely to retrieve it. At the same time, it rarely retrieves irrelevant tables.

The importance of subject columns: In our experiments we considered labels on any columns in the tables, but we observe the importance of subject columns in two ways. First, in 80.16% of the results returned by TABLE, the class label was found in the subject column. For the other ∼20%, we typically observed tables that had more than one possible subject column. Second, in our collection of 168 tables for which we know the subject column and the binary relations, we observed the following. Of the pairs of columns that involved a subject, our algorithms found labels in 43.3% of the cases, compared to only 19.1% for pairs of arbitrary columns.

5. RELATED WORK

Like us, Limaye et al. [17] annotate tables on the Web with column and relationship labels. However, unlike us, their goal is to choose a *single* label from an ontology (YAGO [26]). They propose a graphical model for labeling table columns with types, pair of columns with binary relations, and table cells with entity IDs, and use YAGO as a source for their labels. The key idea of that work is to use joint inference about each of the individual components to boost the overall quality of the labels. As we show in our experiments, YAGO includes only a small fraction of the labels we find. In particular, YAGO includes less than 100 binary relationships, hence our work is the first that tries to detect binary relationships at any serious scale. In principle, we can also apply joint inference with our techniques as the component inference methods, but we leave that for future work.

Cafarella et al. [6] considered the problem of table search, but approached it as a modification of document search. They added new signals to ranking documents, such as hits on the schema elements and left-hand columns. The weights of the new signals were determined by machine learning techniques. As we show in Section 4, table search based aided by our annotations offers significantly higher precision than that of [6].

Several works have considered how to extract and manage data tables found on the Web (e.g., [5, 11, 15]), but did not consider the

annotation or search problems. Gupta and Sarawagi considered how to answer fact queries from lists on the Web [12]. In addition, there has been a significant body of work considering how to rank tuples within a single database in response to a keyword query [14]. The distinguishing challenge in our context is the vast breadth of the data and the fact that it is formatted on Web pages in very different ways.

Downey et al. [10] proposed a theoretical model for measuring the confidence of extractions from the Web. They proposed a combinatorial "urns" model that computes the probability that *a single extraction* is correct based on sample size, redundancy, and corroboration from multiple extraction patterns. In contrast, we compute the probabilistic distribution of semantic labels for columns in Web tables based on *a set of cell values/pairs*. Hence, one of the challenges in our model is to provide smaller weights for missing extractions when the entities they involve do not appear frequently on the Web. The output of the "urns" model can be used as one of the inputs to our model to infer the label distributions.

Existing methods for extracting classes of instances from the Web require sets of instances that are each either unlabeled [18, 23, 29], or associated with a class label [1, 13, 21, 22, 30]. When associated with a class label, the sets of instances may be organized as flat sets or hierarchically, relative to existing hierarchies such as WordNet [25, 26] or the category network within Wikipedia [24, 31]. To our knowledge, the isA database described in this paper is larger than similar databases extracted from unstructured text. In particular, the number of useful extracted class labels (e.g., class labels associated with 10 instances or more) is at least one order of magnitude larger than in the isA databases described in [27], although those databases are extracted from document collections of similar size, and using the same initial sets of extraction patterns as in our experiments.

Previous work on automatically generating relevant labels, given sets of items, focuses on scenarios where the items within the sets to be labeled are descriptions of, or full-length documents within document collections [8, 9, 28]. Relying on semi-structured content assembled and organized manually as part of the structure of Wikipedia articles, such as article titles or categories, the method introduced in [8] derives labels for clusters containing 100 full-length documents each. In contrast, our method relies on isA relations and binary relations automatically extracted from unstructured text within arbitrary Web documents, and computes labels given textual input that is orders of magnitude smaller, i.e., table columns.

6. CONCLUSIONS

We described algorithms for partially recovering the semantics of tables on the Web. We explored an intriguing interplay between structured and unstructured data on the Web, where we used text on the Web to recover the semantics of structured data on the Web. Since only the breadth of Web matches the breadth of structured data on the Web, we are able to recover semantics effectively. In addition, we give a detailed breakdown of when our techniques will not work and how these limitations can be addressed.

In current work, we are investigating better techniques for open information extraction in order to recover a larger fraction of binary relationships and techniques for recovering numerical relationships (e.g., population, GDP). The other major direction of research is increasing our table corpus by extracting tables from lists [11] and structured websites and PDF files.

7. REFERENCES

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, pp 2670–2676, 2007.

[2] M. Banko and O. Etzioni. The Tradeoffs Between Open and Traditional Relation Extraction. In *ACL*, pp. 28–36, 2008.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, pp. 144–152, 1992.

[4] T. Brants. TnT — A Statistical Part of Speech Tagger. In *ANLP*, pp. 224–231, 2000.

[5] M. Cafarella, A. Halevy, and N. Khoussainova. Data Integration for the Relational Web. *PVLDB*, 2(1):1090–1101, 2009.

[6] M. Cafarella, A. Halevy, D. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the Power of Tables on the Web. *PVLDB*, 1(1):538–549, 2008.

[7] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Uncovering the Relational Web. In *WebDB*, 2008.

[8] D. Carmel, H. Roitman, and N. Zwerding. Enhancing Cluster Labeling Using Wikipedia. In *SIGIR*, pp. 139–146, 2009.

[9] D. Cutting, D. Karger, and J. Pedersen. Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections. In *SIGIR*, pp. 126–134, 1993.

[10] D. Downey, O. Etzioni, and S. Soderland. A Probabilistic Model of Redundancy in Information Extraction. In *IJCAI*, pp. 1034–1041, 2005.

[11] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting Relational Tables from Lists on the Web. *PVLDB*, 2:1078–1089, 2009.

[12] R. Gupta and S. Sarawagi. Answering Table Augmentation Queries from Unstructured Lists on the Web. *PVLDB*, 2(1):289–300, 2009.

[13] M. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *COLING*, pp. 539–545, 1992.

[14] P. Ipeirotis and A. Marian, editors. *DBRank*, 2010.

[15] Z. G. Ives, C. A. Knoblock, S. Minton, M. Jacob, P. P. Talukdar, R. Tuchinda, J. L. Ambite, M. Muslea, and C. Gazen. Interactive Data Integration through Smart Copy & Paste. In *CIDR*, 2009.

[16] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *IJCAI*, pp. 729–737, 1997.

[17] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. In *VLDB*, pp. 1338–1347, 2010.

[18] D. Lin and X. Wu. Phrase Clustering for Discriminative Learning. In *ACL-IJCNLP*, pp. 1030–1038, 2009.

[19] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[20] M. Paşca. The Role of Queries in Ranking Labeled Instances Extracted from Text. In *COLING*, pp. 955–962, 2010.

[21] M. Paşca and B. Van Durme. Weakly-Supervised Acquisition of Open-Domain Classes and Class Attributes from Web Documents and Query Logs. In *ACL*, pp. 19–27, 2008.

[22] P. Pantel and M. Pennacchiotti. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In *COLING-ACL*, pp. 113–120, 2006.

[23] M. Pennacchiotti and P. Pantel. Entity Extraction via Ensemble Semantics. In *EMNLP*, pp. 238–247, 2009.

[24] S. Ponzetto and R. Navigli. Large-Scale Taxonomy Mapping for Restructuring and Integrating Wikipedia. In *IJCAI*, pp. 2083–2088, 2009.

[25] R. Snow, D. Jurafsky, and A. Ng. Semantic Taxonomy Induction from Heterogenous Evidence. In *COLING-ACL*, pp. 801–808, 2006.

[26] F. Suchanek, G. Kasneci, and G. Weikum. YAGO: a Core of Semantic Knowledge Unifying WordNet and Wikipedia. In *WWW*, pp. 697–706, 2007.

[27] P. Talukdar, J. Reisinger, M. Paşca, D. Ravichandran, R. Bhagat, and F. Pereira. Weakly-Supervised Acquisition of Labeled Class Instances using Graph Random Walks. In *EMNLP*, pp. 582–590, 2008.

[28] P. Treeratpituk and J. Callan. Automatically Labeling Hierarchical Clusters. In *DGO*, pp. 167–176, 2006.

[29] R. Wang and W. Cohen. Iterative Set Expansion of Named Entities Using the Web. In *ICDM*, pp. 1091–1096, 2008.

[30] R. Wang and W. Cohen. Automatic Set Instance Extraction using the Web. In *ACL-IJCNLP*, pp. 441–449, 2009.

[31] F. Wu and D. Weld. Automatically Refining the Wikipedia Infobox Ontology. In *WWW*, pp. 635–644, 2008.

APPENDIX

A. TYPICAL CHALLENGES POSED BY TABLES ON THE WEB

There are billions of HTML tables on the Web, but the vast majority of them use the HTML table construct for formatting purposes. Even if we consider only tables that contain high-quality data that we would imagine storing in a database, it is still not the case that they are all formatted as sets of rows and columns. The variation in formatting is one of the key reasons that table search is hard. The following examples illustrate some of the typical challenges posed by tables on the Web. At the core, the reasons all these challenges arise is because tables on the Web are formatted for human consumption, not machine consumption and that each table is constructed with a different set of assumptions and context in mind.

Figure 3 shows a vertical table, where the table is transposed so the column names are actually all in the first column of the table, and the column to its right corresponds to a row. Such tables are extremely common on the Web and differentiating between them and ordinary tables can be very tricky.

Figure 4 shows a relatively well structured table. However, the main challenge is to figure out what relation this table is representing. In this case, the table stores the winners of the men's Boston marathon, but that can only be gleaned by reading the text preceding the table. There is no standard location where the table name appears.

Figure 5 shows a table where some of the rows are sub-headers for the rows following them. This is very common when tables are formatted for human consumption (as they would be in spreadsheets). One of the challenges raised by this practice is that the type of the column may not be uniform because the sub-headers will not be of the same type as the other rows, or that one of the rows may be an aggregate of some of the rows preceding it.

The other main challenges concern the vast heterogeneity in attribute names and the fact that they are often long and span multiple rows at the top of the table. In addition, some tables have an important selection condition that is not explicit in the table. For example, we may have a table describing the coffee production of various countries, but the fact that this is the production for the year 2006 is only mentioned in the text.

B. DETAILS ON RECOVERING BINARY RELATIONS

In this section, we provide additional details on the treatment of binary relations.

TextRunner: TextRunner uses a Conditional Random Field (CRF) for the detection of relations. First, it chunks all phrases in the Web documents of the corpus and identifies noun phrases, which are then treated as candidate entities of the extraction. The CRF used takes into account a number of features, such as part-of-speech tags (predicted using a separately trained maximum-entropy model), regular expressions (e.g., detection of capitalization, punctuation), context words and conjunctions of features occurring within six words to the right or left of the current word. Using these features, TextRunner makes a prediction about whether or not a relation has been detected; if one has been detected, then a tuple is inserted into the relations database. As an example, TextRunner is able to extract the tuple (Microsoft, is headquartered in, Redmond) from the phrase Microsoft is headquartered in beautiful Redmond.

The formal model for binary relations: A similar derivation applies for binary relations. Suppose, $R = \{r_1, \ldots, r_n\}$ is the set of value pairs that occur in two columns A and B, and $b_1, \ldots, b_m$ is the set of binary relations, then

$$B(b_i, R) = K_b \times \prod_j \frac{\Pr[b_i \mid r_j]}{\Pr[b_i]}, \text{ such that } \sum_i B(b_i, R) = 1.$$

Each of the conditional probabilities $\Pr[b_i \mid r_j]$ and the prior probability $\Pr[b_i]$ are estimated using the relations database. The score $Score(b_i, r_j)$ in this case corresponds to the number of TextRunner extractions of the binary relation. Equations 3 and 4 can be

Paper Number	20206-PA
DOI What's this?	10.2118/20206-PA
Title	**Theoretical Study of Water Blocking in Miscible Flooding**
Authors	Muller, Thomas, BEB Erdgas and Erdol GmbH; Lake, Larry W., U. of Texas
Journal	SPE Reservoir Engineering
Volume	Volume 6, Number 4
Date	November 1991
Pages	445-451
Copyright	1991. Society of Petroleum Engineers
Language	English

Figure 3: A vertical table: the column names appear in the first column, and each other column corresponds to a database row.

Men's Open

Year	Athlete	Country/State	Time	Notes
1897	John J. McDermott	United States (NY)	2:55:10	course record
1898	Ronald J. MacDonald	Canada	2:42:00	course record
1899	Lawrence Brignolia	United States (MA)	2:54:38	
1900	John "Jack" Caffery	Canada	2:39:44	course record
1901	John "Jack" Caffery	Canada	2:29:23	2nd victory, course record
1902	Sammy Mellor	United States (NY)	2:43:12	
1903	John Lorden	United States (MA)	2:41:29	
1904	Michael Spring	United States (NY)	2:38:04	
1905	Frederick Lorz	United States (NY)	2:38:25	
1906	Tim Ford	United States (MA)	2:45:45	
1907	Thomas Longboat	Canada	2:24:24	course record
1908	Thomas Morrissey	United States (NY)	2:25:43	
1909	Henri Renaud	United States (NH)	2:53:36	
1910	Fred Cameron	Canada	2:28:52	
1911	Clarence DeMar	United States (MA)	2:21:39	course record

Figure 4: The relation represented by the table appears in the text surrounding the table.

PLANT	COLOR	HEIGHT	BLOOM PERIOD
SHRUBS			
Azalea	variable	shrub	spring
Buddleia	blue, pink, white	shrub	midsummer-fall
Lilac	lavender, white, pink	shrub	spring
Sumac	white	shrub	spring
Vaccinium spp.	white, pink	low shrubs	spring-early summer
Viburnums	white	shrubs	spring
CULTIVATED ANNUALS			
Alyssum	violet, white	4 inches	summer-fall
Candytuft	white, pink	8-10 inches	spring-summer
Cosmos	white, lilac, red, yellow	1-3 feet	late summer

Figure 5: Some of the rows in the table are sub-headers rather than actual rows.

applied as is to estimate conditional and prior probabilities for the binary relation labels.

Extensions to more than binary relations: Although we have focused on binary relations until now, our model can be used for unary, binary, or generally n-ary relation. The unary relation is for example the assignment of a class label to the subject column, the binary relation is for example the assignment of a TextRunner predicate to a pair of columns (one of which is the subject column) and so on.

Usefulness of binary relations in table search: During the table search experiment, we focused on whether schema tokens contained the property P of a (C, P) query. Another approach we could have taken is to let P match on of the assigned binary relations also. We observed for the 100 queries we evaluated, we would have retrieved 4% more tables if we had considered the assigned binary relations.

C. SUBJECT COLUMN DETECTION

Many tables on the Web provide the values of properties for a set of instances (e.g., GDP of countries). In these tables we typically have one column that stores the names of the instances. Our goal in this section is to identify this column, which we refer to as subject column. Before we describe our algorithm, we mention a few caveats. First, the subject column *need not be a key of the table* and may well contain duplicate values. Second, it is possible that the subject of the table is represented by more than one column, and we do not attempt to identify these cases (we also observed that they are relatively very rare). Third, there are many tables that do not have a subject column, and our algorithm will still assign one to them. We will see later that assigning subject columns to tables that do not really have them does not impede search quality.

We consider two algorithms. The first algorithm is based on a simple rule: we scan the columns from left to right. The first column that is not a number or a date is selected as the subject column. The second algorithm is a bit more involved, and is based on learning a classifier for subject columns using support vector machines (SVM) [3].

We model subject detection as a binary classification problem. For each column in a table, we compute features that are dependent on the name and type of the column and the values in different cells of the column. Given a set of labeled examples (i.e., tables and their subject column), we train a classification model that uses the computed features to predict if a column in a table is likely to be subject column. Our algorithm takes care of overfitting, by taking a small number of features into account. As we show in our experiments, while even the simple rule-based approach achieves an accuracy of 83%, we are able to improve that to 94% using our SVM. While we could have experimented with other classifiers, SVMs are generally believed to be more robust to overfitting. Further, SVMs have been shown to work well even with unbalanced training data — in our case, subject columns are far fewer than non-subject columns.

Very briefly, an SVM attempts to discover a plane that separates the two classes of examples by the largest margin. A kernel function is often applied to the features to learn a hyperplane that might be non-linear in the original feature space. We used the radial basis function in our experiments.

A subset of the 25 features we used can be seen in Table 4. While we could have used any number of features that we could construct, using all of them can result in overfitting. To avoid this, we used the following process to identify a small subset of the features that is likely to be sufficient in predicting the subject column. From our training data, we measured the correlation of each of our features with the labeled prediction (is the column a subject). The features were then sorted in decreasing order of correlation. For each value of k, we consider the top-k features (in order of correlation) and trained the SVM classifier. We used n-fold cross validation, i.e., dividing the training set into n parts, and performing n runs where for each run, we trained on $(n-1)$ parts and tested on one. We measured accuracy as the fraction of predictions (subject or not) that are correct for the columns in the test set.

No.	Feature Description
1	**Fraction of cells with unique content**
2	**Fraction of cells with numeric content**
3	Average number of letters in each cell
4	Average number of numeric tokens in each cell
5	**Variance in the number of date tokens in each cell**
6	Average number of data tokens in each cell
7	Average number of special characters in each cell
8	**Average number of words in each cell**
9	**Column index from the left**
10	Column index excluding numbers and dates

Table 4: Subset of the 25 features used in the classification of columns

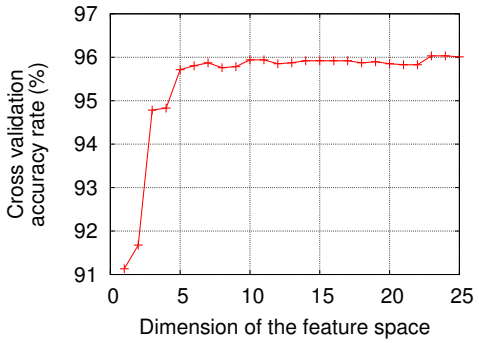

Figure 6: Cross validation accuracy rate.

Figure 6 shows the average cross-validation accuracy as we increase the number of features k. As can be seen, there is not much increase in accuracy for $k > 5$. At the same time, we also found that the number of support vectors in the learned hypothesis decreases for $k \leq 5$ and then starts to increase (a sign of overfitting). Thus, we can identify the set of 5 features that we then use in the rest of our study.

The selected subset of 5 features are bold-faced in Table 4 (1, 2, 5, 8, 9). Not surprisingly, some of them coincide with the baseline hand-crafted rule from our first algorithm.

The SVM classifier, when applied on a new table, can identify more than one column to be the subject (since it is a binary classifier). However, in practice there is typically only one subject column in a table. Hence, we adapt the result of the SVM as follows. Rather than simply using the sign of the SVM decision function, we instead select the column that has the highest value for the decision function.

C.1 Identifying subject columns

We first tested the subject-detection algorithm on tables that were known to have subjects. We manually labeled a set of 1,200 tables from our corpus. The set of tables were separated into a training set of 1,000 tables and a test set of 200 tables. The training set included a total of 4,409 columns of which 1,028 were subject columns. We used 4-fold cross validation to select the 5 most

Sample of Cells from Table Column	Top Class Labels Assigned to Table Column
{Admiral Benbow Inn, Comfort Inn Destin, Country Inn And Suites, Days Inn, Hampton Inn Destin,..}	[hotels, brands, hotel brands, hotel chains, franchises, chains]
{H, He, Ni, F, Mg, Al, Si, Ti, Ar, Mn, Fr, ..}	[elements, trace elements, systems, metals, metal elements, metallic elements, heavy elements, additional elements, metal ions, ..]
{Rose Macaulay, Dorothy Sayers, Graham Greene, Matthew Arnold, A.E. Housman, ..}	[authors, writers, figures, literary figures, poets, favorite authors, famous authors, famous people, british authors, contemporaries, ..]
{Keil Software, Byte Craft Limited, BKR Software, Razorcat, Infineon Technologies, ..}	[companies, semiconductor companies, customers, manufacturers, semiconductor manufacturers, technology companies, clients, vendors, suppliers, ..]
{256Mb, 512Mb, 1Gb, 2Gb, ..}	[capacities, storage capacities, sizes, available capacities, memory configurations, memory sizes, storage capacity memory cards, ipods, memory card storage capacities, data storage capacities, drives, ..]
{Acute Otitis Media Infection, Bronchitis, Proteus Urinary Tract Infection, Streptococcal Tonsillitis, ..}	[infections, common infections, bacterial infections, respiratory infections, respiratory symptoms, acute respiratory infections, respiratory tract infections,..]
{Muscular-Skeletal, Digestion, Nervous, Circulation, Respiration, Reproduction, Excretion, Symmetry, ..}	[systems, processes, physiological processes, biological processes, physiological systems, organ systems, life processes, body systems, vital processes, physiologic processes, bodily systems, factors, properties, metabolic processes, ..]

Table 5: Examples of ranked lists of class labels attached by run $\mathbf{R_{10}}$ to various table subject columns.

Class Name	Property Names	Class Name	Property Names
presidents	political party, birth	amino acids	mass, formula
antibiotics	brand name, side effects	apples	producer, market share
asian countries	gdp, currency	australian universities	acceptance rate, contact
infections	treatment, incidence	baseball teams	colors, captain
beers	taste, market share	board games	age, number of players
breakfast cereals	manufacturer, sugar content	broadway musicals	lead role, director
browsers	speed, memory requirements	capitals	country, attractions
cats	life span, weight	cereals	nutritional value, manufacturer
cigarette brands	market share, manufacturer	clothes	price, brand
constellations	closest constellation, date discovered	countries	capital, gdp
laptops	cpu, price	diseases	incidence, risks, mortality
dogs	life span, weight	dslr cameras	price, megapixels
erp systems	price, manufacturer	eu countries	year joined, currency
european cities	population, country	external drives	capacity, manufacturer
extreme sports	web sites, events	food	calories, type
football clubs	year founded, city	french speaking countries	gdp, population
games	age, platform	google products	launch date, reviews
greek cities	location, main attractions	guitars	manufacturer, price
healthy food	nutritional value, cost	hormones	effects
household chemicals	strength, risks	hurricanes	location, date
inventors	invention, birth	irish counties	population, area
jewish festivals	date, origin	lakes	depth, altitude
law firms	partners, address	macintosh models	cost, release date
maryland counties	zip code, population	mobile phones	weight, operating system
movie stars	income, awards	names	popularity
nba stars	team		

Table 6: Query set for the user study: Each query consists of a class name and a single property name.

relevant column features. Recall that our SVM algorithm always assigns *some* subject column.

On the test set of 200 tables, we found that the simple hand-crafted rule identified the subject column in 83% of the tables. Using the SVM method we were able to identify the subject column in 94% of the tables.

We then tested the subject detection algorithm on an arbitrary set of tables. We collected a separate random sample of 160 tables, 63 of which had no subject columns. We found that of the 97 tables that do have a subject column, we identify it correctly in 92 cases. Thus, we achieve an overall precision of $\frac{92}{160} = 0.58$ and a recall of $\frac{92}{97} = 0.95$.

We note that the lower precision we incur by marking a subject column in every table is not a significant penalty to the eventual precision of table search. This is because the subject column is exploited in the table search algorithm only if it is also assigned a class label. If a column were not a true subject column, it is unlikely to be assigned a class label. This is substantiated in our training data: class labels were assigned for only one of the 68 $(= 63 + (97 - 92))$ incorrectly marked subject columns, whereas labels were assigned to 23 out of the 92 columns that were correctly identified.

C.2 Labeling subject columns

We assigned class labels to the subject columns of a set of tables using the Majority algorithm with parameter t set to 10% (for details see Section 4.1.1). Some examples of these assignments are shown in Table 5.

D. FULL LIST OF QUERIES

The four approaches TABLE, DOCUMENT, GOOG, and GOOGR were compared in Section 4.2 on a query set of 100 class-property queries. The class names were extracted from an analysis of the query logs of Google Squared. The property names were contributed by the authors. Table 6 lists the complete set of class and property names that were used in the comparison.