Table Search Using Recovered Semantics

Petros Venetis[*] , Alon Halevy, Jayant Madhavan, Marius Paşca,
Warren Shen, Fei Wu, Gengxin Miao[†] , Chung Wu
Google Inc.
{petros, halevy, jayant, mars, whshen, wufei, gxmiao, chungwu}@google.com

ABSTRACT

We consider the problem of searching for tables in a large table corpus. The Web offers a corpus of 100 million tables, and smaller but sizable corpora are found within enterprises or individual repositories (e.g., data.gov). Table search is challenging because the semantics of the data are typically not explicit in the table itself, and signals that work well for search over document corpora do not apply as well to table corpora.

We describe the TableFinder system that partially recovers the semantics of the tables in the corpus, by mapping tables into a database of class labels that is automatically extracted from the Web itself. The database of classes has very wide coverage, but is also noisy. TableFinder identifies a column in each table corresponding to the table's subject and identifies the classes describing the values in that column. Query answering proceeds by considering tables whose class labels and properties are relevant to the query. We describe experiments that illustrate that TableFinder provides much higher accuracy compared to approaches that extend document search, and we characterize what fraction of tables on the Web can be annotated using our approach.

1. INTRODUCTION

We consider the problem of searching for data sets in a large table corpus (e.g., *country population*, or *dog breeds life span*). Table corpora are found in several contexts. The Web offers the largest corpus of tables with over 100 million tables about a broad spectrum of topics [6, 12]. Similarly, many enterprises have internal collections of tables (e.g., data.gov). In addition, cloud-based data management products, e.g., Google Fusion Tables [11], Factual (www.factual.com), Socrata (www.socrata.com), and Swivel (www.swivel.com), enable users to upload and publish tables. In all of these contexts, exploration of the data begins with search. Furthermore, search is necessary to make data sets easily discoverable, and therefore provide incentives for data owners to publish their data. Finally, table search is a first step towards answering queries that combine data from multiple tables in a corpus.

[*]Visiting from Stanford University.

[†]Visiting from U.C. Santa Barbara.

Table search is challenging because the semantics of the data are typically not explicit in the table itself. To support search over a collection of tables, the system needs to be aware of these semantics. 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. Attribute names can vary considerably even when modeling the same underlying data, and two tables may look very similar to each other, but have very different meanings. For example, the table for coffee production by country in 2006 looks the same as that for 2010, but the year may appear anywhere in the text surrounding the table. As a result, signals that work well for search over document corpora 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. On the Web, table search is further complicated by the problem of identifying which HTML tables contain good content, rather than merely providing a formatting convenience.

In principle, we would like to associate semantics with each of the tables in the corpus, and use the semantics to guide the retrieval and ranking of search results. However, we do not want to rely on hand-coded domain knowledge which would not scale to the breadth of topics and heterogeneity we witness in table corpora. Instead, we describe the TableFinder System that leverages an isA database that is automatically extracted from the Web. The isA database contains a set of pairs of the form (instance, class), and has very broad coverage of topics, but is also noisy. TableFinder recovers the semantics of tables by trying to capture the class of entities the table is about.

We recover the semantics of a table by labeling some of its columns with classes in the isA database. If a substantial fraction of the cells in a column A are labeled with a class C in the database, we add C to the labels of the column A. To ensure that these labels capture the semantics of the table, we determine which columns are likely to be the *subject* of the table, and only consider the labels of those columns. We show that the labels we generate describe the contents of the table well and are rarely explicit in the table itself. As a result, we can (1) provide table search with much higher precision using the recovered semantics and (2) more effectively determine which HTML tables contain useful content. Furthermore, we characterize the cases in which we cannot recover the semantics of tables using our approach.

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 from unstructured data on the Web to recover semantics of structured data on the Web.

• We describe algorithms for recovering semantics of tables: (1) an algorithm for determining the subject column of an arbitrary table, and (2) an algorithm for automatically associating a set of classes with a given table.

• We describe an experimental evaluation of TableFinder 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 show that it is an order of magnitude more than is possible by mapping tables to Wikipedia classes, 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.

Section 2 defines our terminology and the table search problem. Section 3 describes how we recover semantics of tables, and Section 4 describes how we use the semantics to perform table search. Section 5 describes our experimental evaluation. Section 6 describes related work, and Section 7 concludes.

2. PROBLEM SETTING

We begin by describing our problem setting.

Table corpus: In our corpus each table is a set of rows, and each row is a sequence of cells with data values (see Figure 1 for an example). In table corpora extracted from the Web, the tables themselves may be semi-structured and their meta-data automatically extracted, and therefore incomplete or incorrect. Specifically:

• We may not have a name for the table (i.e., the relationship 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.

• The values in the rows will typically be of a single data type, but there may be exceptions.

• There may be comment or sub-header rows in the table (as one would typically put in a spreadsheet).

• The quality of tables in the corpus may vary significantly. For example, on the Web it is hard to determine automatically whether a region of text found within the HTML table tags represents a high-quality table or whether the author was using the HTML table construct as a formatting mechanism. Even within tabular data found on the Web, the quality may vary. Appendix A walks through several examples that illustrate the variability of table formatting on the Web.

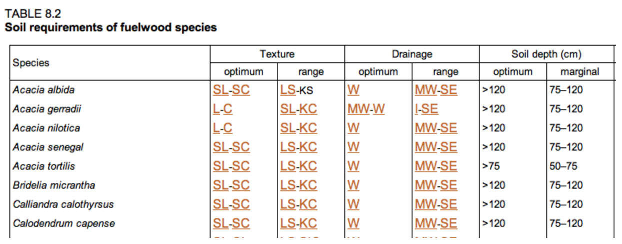


TABLE 8.2
Soil requirements of fuelwood species

Species	Texture		Drainage		Soil depth (cm)	
	optimum	range	optimum	range	optimum	marginal
Acacia albida	SL-SC	LS-KS	W	MW-SE	>120	75–120
Acacia gerradii	L-C	SL-KC	MW-W	I-SE	>120	75–120
Acacia nilotica	L-C	SL-KC	W	MW-SE	>120	75–120
Acacia senegal	SL-SC	LS-KC	W	MW-SE	>120	75–120
Acacia tortilis	SL-SC	LS-KC	W	MW-SE	>75	50–75
Bridelia micrantha	SL-SC	LS-KC	W	MW-SE	>120	75–120
Calliandra calothyrsus	SL-SC	LS-KC	W	MW-SE	>120	75–120
Calodendrum capense	SL-SC	LS-KC	W	MW-SE	>120	75–120

Figure 1: An example table on the Web, describing properties of various tree species. Full table at http://www.fao.org/docrep/009/t0839e/T0839E09.htm

Queries: Ultimately, the most common interaction with a table search module is by posing keyword queries, for two reasons. First, since the tables are very heterogeneous and are about such a vast range of topics (let alone being in over 100 languages), it is unreasonable to expect a schema for querying. Second, in most querying contexts, users will not know the schema for each domain.

We analyzed the Google query stream for queries that could be answered by table search. We found three patterns of queries:

• Find a property of a set of instances or entities (e.g., *coffee production of African countries*).

• Find a property of an individual instance (e.g., *birth date of Albert Einstein*).

• Find a multi-value property of an instance (e.g., *exports of India*).

In order to study the challenges of table search, 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, but we do not consider the problem of transforming an arbitrary keyword query into a pair (C, P) in this paper. The table in Figure 1 shows an answer that TableFinder found for the query *(trees, depth)*.

The second kind of query is addressed in works such as [12, 2, 1]. In that context, the answer is not necessarily found in tables but can also be extracted from free text (and corroborated against multiple occurrences on the Web). The third kind of query can be addressed by extensioning our work.

Answers: In this paper our goal is to return a ranked list of tables in answer to the query. However, our techniques lay the foundation to 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.

Context of table search: There are several contexts in which table search can be considered, and each emphasizes a slightly different aspect of the problem. In this paper we consider the context in which the users are posing a query directly over a corpus of tables, and are therefore explicitly specifying that they are interested in answers that are tables. Ultimately, our goal is to incorporate table search into general Web search. The additional challenge there is identifying *when* the users are asking for tabular data.

Document-based approach: Previous work considered table search as a modification of document search [6]. The approach is based on adding 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. The justification for this approach is twofold. First, it enables us to make relatively slight modifications to the overall web-search infrastructure. Second, when we consider table search in the wild (i.e., with any other search), then the web signals are typically the most important, and they often capture what the table is about in a rather robust way. As we show in Section 5, that approach suffers from low precision compared to ours.

3. RECOVERING SEMANTICS

The key idea underlying our approach is to try to recover some of the semantics of the tables in the corpus, and let the semantics guide search. At the scale and breadth of the corpora we are considering, manually annotating the semantics of tables will not scale. Therefore we automatically recover the semantics leveraging resources that are already on the Web. In particular, as we describe in Section 3.2, we use a set of class labels and instances that are extracted by examining specific linguistic patterns on the Web. Importantly, our goal is not necessarily to recover the most precise semantic descriptions of tables (i.e., we cannot compete with manual labeling), but just enough to provide useful signal for search.

Our method for recovering semantics of tables corresponds to reverse engineering an entity-relationship diagram from a set of tables in the corpus. Specifically, we do the following:

Step 1: We identify in every table a column that is the *subject* of the table. The intention is to identify tables that store properties of a set of instances. Of course, not every table will have a subject column. We describe the algorithm for identifying subject columns in Section 3.1. In the table in Figure 1, the algorithm will deem the first column of the table to be the subject column. While the first column is often the subject column, that is not always the case. Our experiments show that the first column is the correct subject column in 72% of the cases.

Step 2: For every table with a subject column we identify a ranked list of classes in the isA database that describe the instances in the column. We describe the ranking algorithm in Section 3.2. In our running example, the table will get the labels species and trees.

As a result of these two steps, tables now have classes associated with them. Given a query of the form (C, P), we consider tables that match the class C and include the property P. We note that there are other benefits to recovering semantics of tables [17]. First, we can effectively find candidate pairs of tables to join or union. Second, we can look at the attributes that are typically associated with tables in a particular class and find class-specific synonyms for attribute names.

3.1 Detecting the subject column

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 the *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. For example, a table for coffee production by country may have two rows for Brazil, one for each harvesting season. 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 (we describe the details in Appendix B). 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.

3.2 Attaching classes to tables

We attach class labels to tables by mapping the subject column to an isA database, which consists of pairs of the form (instance, class) (e.g., (singapore, southeast asian countries), or (hepatitis, infectious diseases)) mined from text on the Web. We first briefly describe how the isA database is created (see [19] for more details), and then describe how we map the subject column into it.

Creating the isA database: The isA database is extracted from the Web by mining the Web for patterns of the form:

$\langle$[..] C [such as|including] I [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 mine 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, 23].

To construct the isA database, we mined patterns from 100 million documents in English and 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*, *southeast asian countries* and *south 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 [19] is more then 90% and the accuracy for class instances is almost 80%.

To improve the coverage of the database described in [19], 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 cities. The candidate instances are scored across all documents, and added to the list of instances extracted for the class label [27]. This increases coverage with respect to instances, although not with respect to class labels.

Class ranking: Given the candidate matches, we now compute a score for every pair (I, C) using the following formula that counts the number of different patterns the pair occurs in and the frequency of their occurrences.

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

In the formula, $Pattern(I, C)$ is the number 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, after converting punctuation to whitespace and reducing whitespace to a single space. Occurrences of

(I, C) with the same sentence fingerprint are only counted once in $Freq(I, C)$.

Associating classes with tables: Our goal is to determine a set of class labels that describe the instances occurring in the subject column of a table. The isA database can provide a list of class labels for each cell in the subject column. The algorithm shown in Algorithm 1 merges these lists and determines a list of class labels for the column.

The algorithm takes two parameters. The first, C-per-I, is the number of labels to consider for each cell in the column (in our implementation we use 20). The second, t, is the fraction of cells in the column that need to have a label C in order for C to be considered for the label of the column. As we increase t, we will get fewer but more accurate labels. In Section 5.2 we experiment with different values of t.

Algorithm 1: Assignment of classes to table columns.

```
Input      : $IL$, a list of the $L$ cells of a column
             $R$, an isA database
Parameters: C-per-I, number of class labels to retrieve per instance
             $t$, the fraction of cells in the column that need to have
label $C$
foreach cell $I$ in the column do
    $LV(I)$ = empty list;
    if $R$ contains $I$ then
        $LV(I) \leftarrow$ top-(C-per-I) labels assigned to $I$ in $R$ ;
    end
end
return MergeLists($LV$, $t$);
```

Procedure MergeLists(LV, t)

```
foreach class label $C$ appearing in some list of $LV$ do
    count in how many cells it appears in;
end
$S \leftarrow$ {class labels that appear in $\geq t\%$ of the cells of the column} ;
return top-$k$ class labels of $S$ according to $MergedScore(C)$;
```

In the MergeLists step in the algorithm, the per-instance retrieved lists of class labels are merged based on the relative ranks of the class labels within the retrieved lists:

$$MergedScore(C) = \sum_L \frac{1}{Rank(C, L)}$$

where $Rank(C, L)$ is the rank of C in the L^{th} list of class labels computed for the corresponding input instance. The rank is set to 1,000, if C is not present in the L^{th} list. By using the relative ranks of the class labels within the input lists, and not their scores, the outcome of the merging is less sensitive to how class labels of a given instance are scored within the extracted labeled instances. We also demand that an assigned label appears in at least $t\%$ of the cells in the column; in Section 5.2 we give details on how to decide good values of t. Thus, given an input table column, a ranked list of class labels is computed in decreasing order of the merged scores of each class label. In case of ties, the actual scores of the class labels within the extracted labeled instances serve as a secondary ranking criterion.

4. QUERY ANSWERING

Our table search system, TableFinder, leverages the class labels we computed above. Given a query of the form (C, P), where C is a class name and P is a property, it proceeds as follows:

Step 1: Consider tables in the corpus that have the class label C in the top-k labels according to Algorithm 1.[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: occurences of P on the tokens of the schema row, 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 occurence of P in the schema row (which exist in 71% of the tables [7]).

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.

5. EXPERIMENTS

In this section, we evaluate TableFinder and its components. We evaluate our ability to identify tables' subject column (Section 5.1) and the quality of the class labels we assign them and what fraction of tables can be assigned good labels (Section 5.2). In Section 5.3 we compare TableFinder to other table-search methods. We start by describing the corpus of tables that we used in our evaluation.

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 that had relatively 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 million tables. We estimate that this corpus represents about a tenth of the tables worth considering for Web-table search today.

5.1 Identifying subject columns

We first tested the subject-detection algorithm on tables that were known to have subjects. We manually labeled a set of 1200 tables from our corpus. The set of tables were separated into a training set of 1000 tables and a test set of 200 tables. The training set included a total of 4409 columns of which 1028 were subject columns. We used 4-fold cross validation to select the 5 most relevant column features (as described in Appendix B). 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

[1]As an extension, when C is not in the isA database, TableFinder 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 TableFinder.

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, ..]
{Frank Thomas, Albert Belle, Kenny Lofton, Jimmy Key, Kirby Puckett, Julio Franco, David Cone, Cal Ripken, Wade Boggs, Mo Vaughn, ..}	[players, stars, leaguers, names, league stars, baseball players, cards, league players, leagues, athletes, star players, veterans, celebrities, years, veteran players, pitchers, future stars, baseball stars, league baseball players, ..]
{Videotape, Fax, Intranet, E-mail, Radio, Meetings, Phone, Printed Materials, Video Conferencing, ..}	[services, communication services, communications, communication systems, communication tools, communications services, business services, advertising media, news media, office services, internet services, media channels, ..]
{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 1: Examples of ranked lists of class labels attached by run R$_{10}$ to various table subject columns

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.

5.2 Assigning class labels

We now consider one of the main questions concerning our approach: how good are the class labels we assign to tables and to what fraction of tables on the Web can such labels be assigned. We discuss the quality of the labels in Section 5.2.1. In Section 5.2.2 we show that our method labels an order of magnitude more tables than is possible with Wikipedia labels, and in Section 5.2.3 we show that the remaining tables can either be labeled using a few domain specific rules or do not contain useful content.

5.2.1 Label quality

We study the quality of labels assigned by Algorithm 1 with the following parameters. We choose C-per-I, the number of labels we consider per cell in the subject column, to be 20. Labels lower than that in the ranking are typically useless or very general. We ran five experiments, R_t, varying the value of t from 10% to 50%. Recall that t is the fraction of cells in the column that we require a class label to have (hence, R$_{50}$ is the most restrictive experimental run, in that it is expected to assign the fewest class labels). From each run, we kept the top-20 class labels according to the ranking produced by the algorithm. Table 1 illustrates the ranked lists of class labels assigned to various tables by run R$_{10}$.

Gold standard: To create a gold standard, we considered a random sample of tables and removed those who did not have any class labels assigned by run R$_{10}$. We then manually removed tables whose subject columns have been incorrectly identified or do not correspond to any meaningful concept. This results in a set of 99 tables. For each table, we presented to human annotators the union of the class labels attached by any of the five experimental runs to the subject column. The annotators determine the correctness of each class label, by inspecting the table and its source Web page and marking each class 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 michigan counties and southwest michigan counties are marked as *vital*; counties and communities as *okay*; and illinois counties and michigan cities as *incorrect*. In addition, the annotators can manually enter any additional class 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 99 tables with an average of 2.6 *vital* and 3.6 *okay* class labels, with 1.3 added manually by the annotators.

Evaluation methodology: For a given table, the actual evaluation consists of automatically comparing the ranked lists of class labels produced by an experimental run to the class labels available in the gold standard. To compute precision, a retrieved class label is assigned a score of 1.0, if it was marked as *vital* or manually added to the gold standard; 0.5, if it was marked as *okay*; or 0.0, otherwise. Comparatively, recall is computed by considering a class label as relevant (score 1.0) if it was marked as *vital* or *okay* or was manually added to the gold standard, or irrelevant (score 0.0) otherwise.

Results: Table 2 summarizes the results. The different rows correspond to selecting the precision and recall for the top-k labels attached to a table. The columns correspond to different experimental runs. As the runs grow gradually more restrictive, the number of tables is smaller (because less tables get labels), varying from 99 for R$_{10}$ down to 45 for R$_{50}$; the precision increases, from 0.30 (run R$_{10}$) to 0.45 (run R$_{50}$) at rank 20; and recall, and the number of tables with any class labels assigned, decrease.

Based on these results we decided to use R$_{50}$ in TableFinder, and hence trade recall for precision for several reasons. First, without agreement on a class label from a majority of rows, the class label may be incorrect. For example, county names and city names often collide, and we must take into account enough names that do not collide to differentiate between the counties and cities labels. Second, recall that the class labels are extracted from free text on the Web. If a label is agreed upon by most rows, then it is not only a correct, but also the common way for people to refer to the set of instances. Users will then be more likely to use this label to search for these instances as well. Finally, if a table result is presented

	Precision (P) and Recall (R) over Subsets (S) of Tables with Assigned Class Labels														
	R_{10}			R_{20}			R_{30}			R_{40}			R_{50}		
Rank (k)	S	P	R	S	P	R	S	P	R	S	P	R	S	P	R
1	99	0.82	0.16	82	0.81	0.15	70	0.88	0.15	55	0.87	0.14	45	0.88	0.12
3	99	0.65	0.44	82	0.62	0.34	70	0.72	0.35	55	0.73	0.32	45	0.71	0.29
5	99	0.55	0.60	82	0.52	0.47	70	0.62	0.46	55	0.64	0.43	45	0.60	0.41
7	99	0.48	0.71	82	0.46	0.53	70	0.56	0.52	55	0.57	0.49	45	0.53	0.47
10	99	0.40	0.80	82	0.40	0.59	70	0.51	0.58	55	0.51	0.55	45	0.48	0.53
15	99	0.33	0.88	82	0.36	0.64	70	0.46	0.62	55	0.47	0.58	45	0.45	0.55
20	99	0.30	0.95	82	0.33	0.65	70	0.44	0.62	55	0.46	0.58	45	0.45	0.55

Table 2: Comparative performance of various experimental runs (R_{10} through R_{50}) when considering the top-k class labels computed over (variable) subsets of tables from the gold standard for which some class labels are assigned by each run (S=size of subset of tables, P=precision, R=recall)

to the user because of an incorrect label, that result is likely to be jarring and useless. Therefore, for class labeling, we prefer to err on the side of caution.

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.

5.2.2 *Labels from ontologies*

Next, we compare the coverage of our labeling to what can be obtained by using a manually created ontology — namely Wikipedia. Currently, the state-of-the-art, precision-oriented isA database is YAGO [24], which is based on Wikipedia. Table 3 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 had at hand), while YAGO assigns labels to ∼185 thousand tables (an order of magnitude difference). This can be 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
Labeled subject columns	1,496,550	185,013
Instances	155,831,855	1,940,797

Table 3: Comparing our isA database and YAGO

We also compared our labeling on the same 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 Algorithm 1. We note that using YAGO will result in higher precision. For the goal of table search though, coverage (and hence recall) is key.

5.2.3 *The unlabeled tables*

Our methods assigned class labels to approximately 1.5 million tables out of the 12.3 million tables in our corpus. When we also considered labels for non-subject columns, we labeled 4.3 million tables. Hence, we investigated why the other tables were not labeled, and most importantly, whether we are we missing any 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 4 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
Unuseful		4	32.52

Table 4: 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 2 for an example). 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 are relevant to (C, P) 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. Common categories include comments on social networks, bug reports and job postings. In addition, we found that many of these tables have enough text in them to be retrieved when relevant. For example, we found course description tables, but they typically have the course number and university on the page. A user would typically search for the course name, rather than via a (C, P) query.

Hence, in summary, though we have annotated about a sixth of our tables, our results suggest that these are the *useful* tables on the Web for (C, P) queries, and therefore that semantic annotation is also a good method for identifying the good tables from the useless ones.

5.3 Table search

We now examine the quality of table search with TableFinder (henceforth referred to as TABLE) by comparing it with three other methods: (1) GOOG: the results returned by 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 considers several signals extracted from the document in the ranking. Specifically, in addition to the document signals considered by TABLE, the document-based algorithm considers 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 of Google Squared [1]. Google Squared allows users to enter the name of a class, for which it then attempts to construct a table with more information about instances of that class. 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 C for a complete listing).

Evaluation methodology: We performed a user study to compare the results of each algorithm. 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 *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 5. 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:

$$P^a(m) = \frac{N_q^a(m)}{N_q(m)} \quad R^a(m) = \frac{N_q^a(m)}{N_q^a(*)}$$

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 5. 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 5, TABLE has the highest precision (0.79 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 much lower, if we only consider the *right on* ratings (0.42). If we only consider queries for which the relevant information was eventually found in a table, TABLE has both the highest precision (0.79) and highest recall (0.62) 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 TableFinder 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.

For this experiment we allowed TABLE to look at the labels assigned to all columns (not just the subject column), in order to increase recall. Even though TABLE was given this flexibility, in 80.16% of the results it returned the class label was found in the subject column, a fact that emphasizes the importance of subject columns. For the other ~20%, we typically observed tables that had more than one possible subject column. Furthermore, the experiments suggest that our ranking of class labels is effectively. In 47% of the cases, the query's class label was the top-1 label assigned by our Algorithm 1, and in 85% of the cases, it belonged in the top-5 labels assigned.

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.

6. RELATED WORK

Cafarella et al. [6] were the first to consider search over a Web-scale corpus of tables and proposed the document-based approach to search which we improved upon. In addition, [6] showed how synonyms for attribute names can be extracted from the schema collection in such a corpus. Limaye, Sarawagi and Chakrabarti [17] propose a graphical model for labeling table columns with types, pairs of columns with binary relations, and table cells with entity IDs, and uses YAGO as a source for their labels. As we showed, we obtain much wider label coverage than YAGO. However, the goal of [17] is to obtain higher precision in order to answer more specific queries: find entities that have a given value for a particular attribute.

Several works have considered how to extract and manage data tables found on the Web (e.g., [15, 5, 10]), but did not consider the search problem. 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.

Existing methods for extracting classes of instances require sets of instances that are each either unlabeled [27, 21, 18], or associated with a class label [13, 20, 2, 19, 28]. When associated with a class label, the sets of instances may be organized as flat sets or

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	**0.63**	**0.77**	**0.79**	0.52	0.51	**0.62**
DOCUMENT	399	24	58	47	93	13	36	32	0.20	0.37	0.34	0.31	0.44	0.50
GOOG	493	63	116	52	100	32	52	35	0.42	0.58	0.37	**0.71**	**0.75**	0.59
GOOGR	156	43	67	59	65	17	32	29	0.35	0.50	0.46	0.39	0.42	0.48

Table 5: 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 5.3.

hierarchically, relative to existing hierarchies such as WordNet [23, 24] or the category network within Wikipedia [29, 22]. 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 [25], 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 [9, 8, 26]. 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 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.

7. CONCLUSIONS

We described TableFinder, a system that partially recovers the semantics of tables in a corpus, and uses the semantics to guide table search. TableFinder uses an isA database extracted from the Web itself, therefore providing very broad coverage of content. Our experiments demonstrate that even with the little semantics we extract, TableFinder achieves very high precision compared to alternative methods and has high recall w.r.t. the tables that are in the corpus. Our experimental results also suggest that TableFinder effectively discovers the useful relational tables on the Web. TableFinder can also determine with high accuracy when there are no relevant results in the corpus, at which point we can fall back on other methods to find non-tabular content.

Extracting classes is just the first step in recovering semantics. The next step is to recover the meaning of the binary (or more) relations represented by columns in a table. Other major directions of research are increasing our table corpus by extracting tables from lists [10], structured websites and PDF files; and extracting hierarchies of attributes to address queries asking for multiple properties (e.g., exports of India). Finally, we are developing methods for parsing a keyword query to determine whether it can be reformulated into a table search query.

8. REFERENCES

[1] Google squared, http://www.google.com/squared.

[2] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the Web. In *IJCAI*, 2007.

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

[4] T. Brants. TnT — a statistical part of speech tagger. In *ANLP*, 2000.

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

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

[10] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the Web. *PVLDB*, 2(1), 2009.

[11] H. Gonzalez, A. Halevy, C. Jensen, A. Langen, J. Madhavan, R. Shapley, and J. Goldberg-Kidon. Google Fusion Tables: Data Management, Integration and Collaboration in the Cloud. In *SIGMOD*, 2010.

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

[13] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 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*, 1997.

[17] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. In *VLDB*, 2010.

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

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

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

[21] M. Pennacchiotti and P. Pantel. Entity Extraction via Ensemble Semantics. In *EMNLP*, 2009.

[22] S. Ponzetto and R. Navigli. Large-scale taxonomy mapping for restructuring and integrating Wikipedia. In *IJCAI*, 2009.

[23] R. Snow, D. Jurafsky, and A. Ng. Semantic Taxonomy Induction from Heterogenous Evidence. In *COLING-ACL*, 2006.

[24] F. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge unifying WordNet and Wikipedia. In *WWW*, 2007.

[25] 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*, 2008.

[26] P. Treeratpituk and J. Callan. Automatically Labeling Hierarchical Clusters. In *DGO*, 2006.

[27] R. Wang and W. Cohen. Iterative Set Expansion of Named Entities Using the Web. In *ICDM*, 2008.

[28] R. Wang and W. Cohen. Automatic set instance extraction using the Web. In *ACL-IJCNLP*, 2009.

[29] F. Wu and D. Weld. Automatically refining the Wikipedia infobox ontology. In *WWW*, 2008.

APPENDIX

A. TYPICAL CHALLENGES POSED BY TABLES ON THE WEB

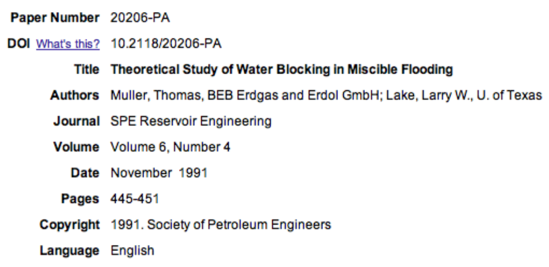
There are billions of HTML tables on the Web, but the vast majority of them use the HTML table construct for formatting purposes. Even if we consider only tables that contain high-quality data that we would imagine storing in a database, it is still not the case that they are all formatted as sets of 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 reason all these challenges arise is because tables on the Web are formatted for human consumption, not machine consumption.

Figure 2 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 3 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.

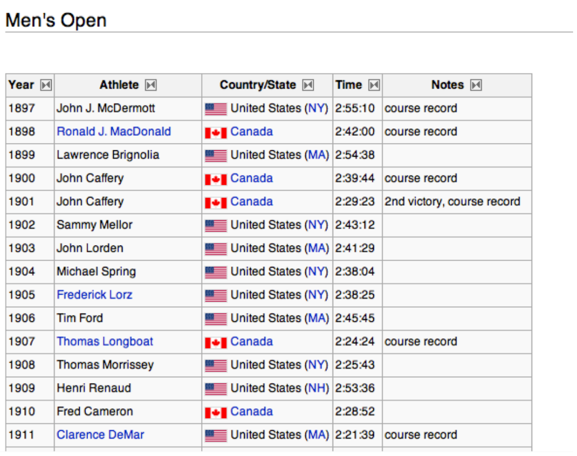
Figure 4 shows a table where some of the rows are sub-headers for the rows following them. This is very common when tables are formatted for human consumption (as they would be in spreadsheets). One of the challenges raised by this practice is that the type of the column may not be uniform because the sub-headers will not be of the same type as the other rows.

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.



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

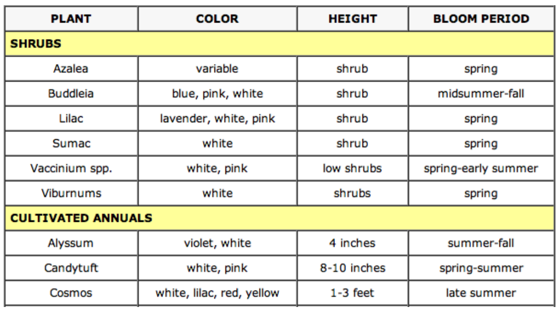
Figure 2: A vertical table: the column names appear in the first column, and each other column corresponds to a database row.



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 Caffery	Canada	2:39:44	course record
1901	John 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 3: 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 4: Some of the rows in the table are sub-headers rather than actual rows.

B. SUBJECT COLUMN DETECTION

Our more advanced algorithm for detecting the subject column considers an SVM classifier to learn a model from our labeled data. 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 6. 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.

Figure 5 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 6 (1, 2, 5, 8, 9). Not surprisingly, some of them coincide with the baseline

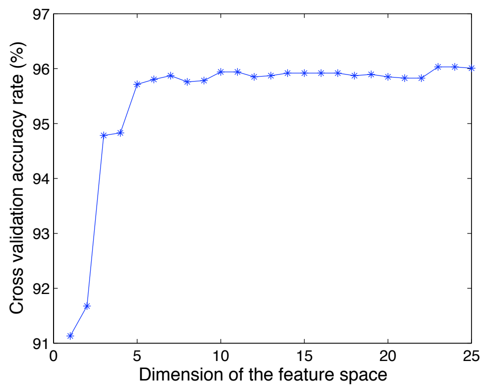


Figure 5: Cross validation accuracy rate.

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. FULL LIST OF QUERIES

The four approaches TABLE, DOCUMENT, GOOG, and GOOGR were compared in Section 5.3 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 7 lists the complete set of class and property names that were used in the comparison.

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

Table 6: Subset of the 25 features used in the classification of 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 7: Query set for the user study: Each query consists of a class name and a single property name.