EntiTables: Smart Assistance for Entity-Focused Tables

Shuo Zhang
University of Stavanger
shuo.zhang@uis.no

Krisztian Balog
University of Stavanger
krisztian.balog@uis.no

ABSTRACT

Tables are among the most powerful and practical tools for organizing and working with data. Our motivation is to equip spreadsheet programs with smart assistance capabilities. We concentrate on one particular family of tables, namely, tables with an entity focus. We introduce and focus on two specific tasks: populating rows with additional instances (entities) and populating columns with new headings. We develop generative probabilistic models for both tasks. For estimating the components of these models, we consider a knowledge base as well as a large table corpus. Our experimental evaluation simulates the various stages of the user entering content into an actual table. A detailed analysis of the results shows that the models' components are complimentary and that our methods outperform existing approaches from the literature.

CCS CONCEPTS

•**Information systems** →**Environment-specific retrieval;** *Users and interactive retrieval; Recommender systems;* Probabilistic retrieval models;

KEYWORDS

Table completion; intelligent table assistance; semantic search

ACM Reference format:
Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proceedings of SIGIR'17, August 07–11, 2017, Shinjuku, Tokyo, Japan.*, , 10 pages.
DOI: http://dx.doi.org/10.1145/3077136.3080796

1 INTRODUCTION

Tables are one of the most effective and widely used tools for organizing and working with data. Spreadsheet programs are among the most commonly used desktop applications, both in business environments and in personal use, because of their ease of use and flexibility. The overall objective of this study is to develop an intelligent personal assistant that can offer smart assistance for people working with tables. It may be imagined as the infamous Office Clippy, albeit we prefer it to be less obtrusive. This study represents the first step towards this ambitious endeavor.

The scenario we consider in this paper is the following. We assume a user, working with a table, at some intermediate stage in the process. At this point, she has already set the caption of the table and entered some data into the table. The table is assumed to have a column header (located above the first content row), which identifies each column with a unique label. We further narrow the focus of our study to tables with an *entity focus*. It means that the leftmost column of the table contains entities. This can also be imagined as having a designated row heading, which may contain only (unique) entities. An entity in the context of this work is a specific object with a unique identifier. (We shall show later in the paper, in §6.2, that a significant portion of tables have an entity focus.) Against this setting, our objective is to aid the user by offering "smart suggestions," that is, recommending (i) additional entities (rows) and (ii) additional column headings, to be added to the table. We shall refer to these tasks as *row population* and *column population*, respectively. See Figure 1 for an illustration.

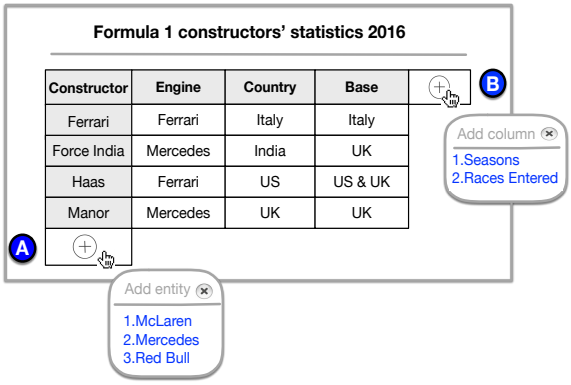

Figure 1: Envisioned user interface. Column headings and the leftmost column are marked with a grey background. The user can populate the table with (A) additional entities (rows) and (B) additional column headings. The suggestions in the pop-ups are updated dynamically as the content of the table changes.

Let us point out here that some elements of these tasks have been addressed in prior work. Our work, however, has not only a different overall motivation, but the specific tasks we tackle have not been addressed in these flavors before. We also introduce a number of innovative elements on the component level.

The task of row population relates to the task of *entity set expansion* [4, 8, 10, 16, 26, 29], where a given set of seed entities (examples) is to be completed with additional entities. We also have a seed set of entities from the leftmost table column. But, in addition to that, we can also make use of the column heading labels and the caption of the table. We show in our experiments, that utilizing these can lead to substantial improvements over using only the seed entities.

The second task, column population, shares similarities with the problem of *schema complement* [2, 8, 12, 30], where a seed table is to be complemented with related tables that can provide additional

SIGIR'17, August 07–11, 2017, Shinjuku, Tokyo, Japan.

DOI: http://dx.doi.org/10.1145/3077136.3080796

columns. Many of these approaches utilize the full table content and also address the task of merging data into the seed table. Here, our focus is only on finding proper column headings, using the same sources as for row population (i.e., leftmost column, header row, and table caption). We show in our experiments that this task can be performed effectively.

In summary, this paper makes the following novel contributions:

- We introduce and formalize two specific tasks for providing intelligent assistance with tables: row population and column population (§3).
- We present generative probabilistic methods for both tasks, which combine existing approaches from the literature with novel components (§4 and §5).
- We design evaluation methodology and develop a process that simulates a user through the process of populating a table with data (§6).
- We perform an experimental evaluation and carry out a detailed analysis of performance (§7 and §8).

All resources developed within this study are made publicly available at http://bit.ly/sigir2017-table.

2 RELATED WORK

There is a growing body of work on web tables and spreadsheets, addressing a range of tasks, including table extension, table completion, table search, table mining, etc. The task of row population is also related to the problem of entity set completion.

Table Extension/Completion. Extending a local table with additional columns based on the corpus of tables is a relatively new research area. The Mannheim Search Joins Engine [12] operates on a corpus of web tables, searches for tabular data describing entities in the local table, and then picks relevant columns from the top-k candidate tables to merge. With a focus on Wikipedia tables, Bhagavatula et al. [2] target column-matched tables with the local table and perform correlation mining to find "interesting" numeric columns. InfoGather [30] is a table augmentation framework based on topic sensitive PageRank for matching the local table against web tables. The context surrounding the tables is leveraged in a machine learning framework, where the similarity between two tables is captured via a set of features. Related tables can be utilized not only for column extension, but for row extension as well. Methods to detect related tables are proposed for the relatedness capture framework in [8]. Two types of table relatedness are identified: entity complement and schema complement. Entity complement tables can be united to produce a meaningful table, and schema complement tables can provide additional meaningful columns. *Table completion* refers to the task of filling missing values in a local table. Ahmadov et al. [1] propose a hybrid data imputation approach, relying on the characteristics of missing values, in order to (i) look up missing values from web data, (ii) predict them using machine learning methods, or (iii) combine both to find the most appropriate values. To look up missing values, two keyword subqueries are created from the input table, to search entities and attributes separately. These resemble our row and column populating subtasks. However, Ahmadov et al. [1] have a different target and merge the two search results for table selection.

Table Search and Mining. There has been an increasing research interest in mining and searching table content, see, e.g., [5, 6, 15, 22, 25, 32]. Wikipedia's tables contain rich, semi-structured encyclopedic content that is hard to query. Muñoz et al. [18] extract factual content from Wikipedia tables in the form of RDF triples, contributing to recovering table semantic and discovering table relations. Apart from the factual content extraction from web tables, *table mining* also covers tasks like *table interpretation* [6, 18, 25] and *table recognition* [7, 33].

Cafarella et al. [6] extracted 14.1 billion HTML tables from a Google crawl, estimating that 154 million of them contain high-quality relational data. The relations extracted from these represent a valuable data resource. To disambiguate web tables, Zwicklbauer et al. [33] propose a methodology to annotate table headers with semantic type information based on the column's content. Similarly, Crestan and Pantel [7] present a supervised framework for classifying HTML tables into their taxonomy. In addition to factual content and relations, numeric attributes are present in a vast number of web tables. However, web tables are not systematic and cannot be used, e.g., for aggregation. To improve the usability of quantities in heterogenous web tables, a line of work aims at detecting quantity mention [9, 11, 14, 21, 24] and canonicalizing table quantities [3, 11, 13, 19]. Another line of work focuses on extracting and fusing numeric attribute values and numeric expressions in natural language text [9, 14, 21, 24].

Tables could be well searched and mined for question answering or for extending knowledge bases. Yin et al. [31] investigate the task of executing queries on knowledge base tables using Neural Enquirer, which is a fully neuralized DNNs model, both for query planning and for query execution. Sekhavat et al. [23] describe a probabilistic method that augments an exiting knowledge base (YAGO). In [28], a table search engine is applied to further expand and enrich Probase, which is a universal probabilistic taxonomy framework capable of understanding the entities, attributes and values in web tables. Knowledge Vault is created as a probabilistic knowledge base [9] by analyzing the extracted content from tabular data, along with other web resources.

Entity Set Completion. The problem of row population is related to the task of *set completion* or *list expansion*, which is to generate a ranked list of entities starting from a small set of seed entities [8]. Bron et al. [4] propose an approach that combines structure-based and text-based similarity between a candidate entity and the seed entities. The QBEES framework [16] is designed as an aspect-based entity model to find similar entities based on one or more example entities. He and Xin [10] focus on entity list data, by picking the top-k entities based on relevance between the candidate entity and seed entities, and then iteratively ranking them according to a combination of relevance and coherence. Similar iterative steps are conducted in [26, 29]. Wang and Cohen [29] use a random walk method for ranking during iterations. Wang et al. [26] focus on web tables, instead of entity lists, with the help of a web table search engine, called WTS.

3 PROBLEM STATEMENT

In this section, we provide a formal description of the tasks we propose to undertake. We refer to Table 1 for our notation.

Table 1: Notation used in this paper.

Symbol	Description
T	Table
c	Table caption
E	Seed entities $E = (e_1, \ldots, e_n)$
L	Seed column labels $L^{(j)} = (l_1, \ldots, l_m)$

Definition 1 (Table): *A table T is grid of cells, which hold values, arranged in $n+1$ rows and m columns. The top row is a special designated place, where the column headings reside. It is followed by n regular (content) rows. We let $L = (l_1, \ldots, l_m)$ be the list of column heading labels. In addition to the grid content, the table also has a caption c.*

Definition 2 (Entity-Focused Table): *A table is said to be entity-focused, if its leftmost column contains only entities as values, and those entities are unique within the column. We let $E = (e_1, \ldots, e_n)$ be the list of entities corresponding to the leftmost table column. I.e., the table takes the following shape:*

$$T = \begin{bmatrix} l_1 & l_2 & \ldots & l_m \\ e_1 & v_{1,2} & \ldots & v_{1,m} \\ e_2 & v_{2,2} & \ldots & v_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ e_n & v_{n,2} & \ldots & v_{n,m} \end{bmatrix},$$

where $v_{i,j}$ ($i \in [1..n], j \in [2..m]$) denote the cell values.

Our objective is to provide intelligent assistance for an user who is working on an entity-focused table. We shall refer to the table that is being edited by the user as *seed table*. We assume that the seed table has already been given a caption, and contains some heading labels (*seed labels*) in the top row and some entities (*seed entities*) in the leftmost column. Note that we do not make any assumptions about the values in the other table cells. Essentially, the $v_{i,j}$ values are immaterial, therefore, we omit them in the followings.[1] When we talk about a table containing entity e, we always mean the leftmost table column.

Our goal is to present suggestions for the user for extending the seed table with (i) additional entities, and (ii) additional column heading labels. Both tasks are approached as a ranking problem: given a seed table, generate a ranked list of entities (for row population) or column labels (for column population).

Definition 3 (Row Population): *Row population is the task of generating a ranked list of entities to be added to the leftmost column of a given seed table, as e_{n+1}.*

Definition 4 (Column Population): *Column population is the task of generating a ranked list of column labels to be added to the column headings of a given seed table, as l_{m+1}.*

[1] We note that the $v_{i,j}$ values may also be utilized for row/column population. However, this is left for future work.

In the following two sections, we present our approaches for row and column population. Following prior studies [1–3, 6–8, 12, 23, 25, 26, 28, 30, 31], we rely heavily on the availability of a large table corpus as an external resource (which, in our case, is extracted from Wikipedia). Additionally, we also exploit information stored about entities in a knowledge base (in our case, DBpedia). Further specifics about our data sources are provided in §6.1.

4 POPULATING ROWS

In this section, we address problem of row population using a two-step approach. We assume that a seed table is given, with a list of n seed entities E, a list of m seed column labels L, and a table caption c. The task is to generate a ranked list of suggestions for entity e_{n+1}, which may be added to the seed table as a new row. First, we identify a set of candidate entities (§4.1), and then rank them in a subsequent entity ranking step (§4.2).

4.1 Candidate Selection

We identify candidate entities using two sources: knowledge base (KB) and table corpus (TC). In the knowledge base, each entity e is described by a set of properties $\mathcal{P}_e$. We focus on two specific properties: types and categories. We discuss these notions in the context of DBpedia, but note that all knowledge bases employ some taxonomy of types. Types in DBpedia are assigned from a small ontology (the DBpedia Ontology, containing a few hundred classes). Categories originate from Wikipedia; these do not form a strict is-a hierarchy, and may be seen more like "semantic sets." Categories are in the order of several 100K. Intuitively, an entity e that has several types or categories overlapping with those of the seed entities represents a good candidate. Thus, we rank entities based on the overlap of these properties, and then take the top-k ones as the set of candidates:

$$score(e, E) = \left| \mathcal{P}_e \cap \left(\cup_{i=1}^{n} \mathcal{P}_{e_i} \right) \right| .$$

When using the table corpus, we search for tables that contain the seed entities or have a similar caption to that of the seed table. This can be efficiently performed using existing retrieval methods against an inverted index of tables. Specifically, we use either the seed table's caption or seed entities as the search query and rank tables using the BM25 retrieval algorithm.

4.2 Ranking Entities

We introduce a probabilistic formulation and rank candidate entities according to the multi-conditional probability $P(e|E, L, c)$. By applying Bayes's theorem and making a full independence assumption between table caption, seed entities, and seed column labels, we factor this probability as follows:

$$\begin{aligned} P(e|E, L, c) &= \frac{P(E, L, c|e)P(e)}{P(E, L, c)} \\ &= \frac{P(E|e)P(L|e)P(c|e)P(e)}{P(E)P(L)P(c)} \\ &\propto P(e|E)P(L|e)P(c|e) \ . \end{aligned} \tag{1}$$

In the last step, we rewrote $P(E|e)$ using Bayes' rule (which cancelled out $P(e)$ and $P(E)$). We further dropped the probabilities $P(L)$ and $P(c)$ from the denominator, since those are the same across all

candidate entities and thus do not influence their ranking. Then, entities are ranked by multiplying (i) the posteriori probability $P(e|E)$ that expresses entity similarity, (ii) the column labels likelihood $P(L|e)$, and (iii) the caption likelihood $P(c|e)$. The reason for keeping the latter two probabilities conditioned on the candidate entity is that column labels and captions are very short. In those cases, the candidate entity offers a richer observation. Below, we discuss the estimation of each of these probabilities.

Note that entities may be ranked using any subset of the components in Eq. (1). We explore all possible combinations in our experimental section (§7). It is our expectation that using all three sources of evidence (seed entities, seed column labels, and table caption) would result in the best performance.

4.3 Entity Similarity

The estimation of $P(e|E)$ corresponds to the task of *entity list completion* (also known as *set/concept expansion* or *query by example*): given a small set of seed entities, complement this set with additional entities. The general idea is to measure the semantic similarity between the candidate entity and the set of seed entities. One line of prior work [4, 16] relies on a knowledge base for establishing this semantic similarity. Another family of approaches [8, 26, 29] leverages a large table corpus for collecting co-occurrence statistics. We combine both these sources in a single model:

$$P(e|E) = \lambda_E P_{KB}(e|E) + (1 - \lambda_E) P_{TC}(e|E) \,, \tag{2}$$

where P_{KB} is based on the knowledge base and P_{TC} is the estimate based on the table corpus.

4.3.1 Estimation Using a Knowledge Base. Bron et al. [4] create a structured entity representation for each entity from the RDF triples describing that entity. The structured representation of an entity is comprised by the set of relations of the entity. Each relation r is modeled as a pair, by removing the entity itself from the triples. E.g., given the triple ⟨`dbr:Japan, dbo:capital, dbr:Tokyo`⟩ describing the entity Japan, the corresponding relation becomes (`dbo:capital, dbr:Tokyo`). We write $\hat{e}$ to denote the structured representation of entity e. Formally, given a set of subject-predicate-object (s, p, o) triples describing the entity (i.e., the entity stands either as subject or object):

$$\hat{e} = \{(p, o) : (s = e, p, o)\} \cup \{(s, p) : (s, p, o = e)\} \,.$$

Similarly, each seed entity is represented as a set of pairs: $\hat{e}_1, \ldots, \hat{e}_n$. The set of seed entities is modeled as a multinomial probability distribution θ_E over the set of relations. The probability $P(e|E)$ is then obtained by considering all relations that appear in the representation of the candidate entity:

$$P_{KB}(e|E) = \sum_{r \in \hat{e}} P(r|\theta_E) = \sum_{r \in \hat{e}} \frac{\sum_{i=1}^{n} \mathbb{1}(r, \hat{e}_i)}{|\theta_E|} \,,$$

where $\mathbb{1}(r, \hat{e}_i)$ is a binary indicator function, which is 1 is r occurs in the representation of $\hat{e}_i$ and is 0 otherwise. The denominator is the representation length of the seed entities, i.e., the total number of relations of all seed entities: $|\theta_E| = \sum_{i=1}^{n} \sum_{r \in \hat{e}_i} \mathbb{1}(r, \hat{e}_i)$.

Instead of using a single model built for the set of seed entities, we also explore an alternative approach by taking the average pairwise similarity between the candidate and seed entities (similar in spirit to [8, 10]):

$$P_{KB}(e|E) \propto \frac{1}{n} \sum_{i=1}^{n} \text{sim}(e, e_i) \,,$$

where $\text{sim}(e, e_i)$ is a similarity function. We consider two alternatives for this function. The first is the *Wikipedia Link-based Measure* (WLM) [17], which estimates the semantic relatedness between two entities based on other entities they link to:

$$\text{sim}_{WLM}(e, e_i) = 1 - \frac{\log(\max(|\mathcal{L}_e|, |\mathcal{L}_{e_i}|)) - \log(|\mathcal{L}_e \cap \mathcal{L}_{e_i}|)}{\log(|\mathcal{E}| - \log(\min(|\mathcal{L}_e|, |\mathcal{L}_{e_i}|)))} \,,$$

where $\mathcal{L}_e$ is the set of outgoing links of e (i.e., entities e links to) and $|\mathcal{E}|$ is the total number of entities in the knowledge base. The second similarity function is the Jaccard coefficient, based on the overlap between the outgoing links of entities:

$$\text{sim}_{Jacc}(e, e_i) = \frac{|\mathcal{L}_e \cap \mathcal{L}_{e_i}|}{|\mathcal{L}_e \cup \mathcal{L}_{e_i}|} \,.$$

4.3.2 Estimation Using a Table Corpus. Another way of establishing the similarity between a candidate entity e and the set of seed entities E is to obtain co-occurrence statistics from a table corpus (as in [1, 8]). We employ a maximum likelihood estimator:

$$P_{TC}(e|E) = \frac{\#(e, E)}{\#(E)} \,,$$

where $\#(e, E)$ is the number of tables that contain the candidate entity together with all seed entities, and $\#(E)$ is the number of tables that contain all seed entities. Provided that the table corpus is sufficiently large, we expect this simple method to provide an accurate estimate.

4.4 Column Labels Likelihood

For computing $P(L|e)$, we consider the tables from the table corpus where the entity appears in the leftmost column. We obtain and combine two different estimates. The first one is the representation of the entity in terms of the words of the column labels, i.e., an unigram language model (LM). The second one is a maximum likelihood estimate using exact label matching (EM), i.e., without breaking labels up to words. We consider each individual label l from the seed column labels and combine the above two estimates using a linear mixture:

$$P(L|e) = \sum_{l \in L} \left(\lambda_L \Big(\prod_{t \in l} P_{LM}(t|\theta_e) \Big) + \frac{(1 - \lambda_L)}{|L|} P_{EM}(l|e) \right) .$$

The first component is a Dirichlet-smoothed unigram language model, calculated using:

$$P_{LM}(t|\theta_e) = \frac{tf(t, e) + \mu P(t|\theta)}{|e| + \mu} \,,$$

where $tf(t, e)$ is the total term frequency of t in column heading labels of the tables that include e in their leftmost column. One may think of it as concatenating all the column heading labels of the tables that include e, and then counting how many times t appears in there. The length of the entity $|e|$ is the sum of all term frequencies for the entity ($|e| = \sum_{t'} tf(t', e)$). The background language model $P(t|\theta)$ is built from the column heading labels of all tables in the corpus.

The exact label matching probability is estimated using:

$$P_{EM}(l|e) = \frac{\#(l,e)}{\#(e)} ,$$

where $\#(l,e)$ is the number of tables containing both e and l, and $\#(e)$ is the total number of tables containing e.

4.5 Caption Likelihood

To estimate the caption likelihood given an entity, $P(c|e)$, we combine two different term-based entity representations: one from the knowledge base and one from the table corpus. Formally:

$$P(c|e) = \prod_{t \in c} \Big(\lambda_c P_{KB}(t|\theta_e) + (1-\lambda_c) P_{TC}(t|e)\Big) .$$

The knowledge base entity representation is an unigram language model constructed from the entity's description (specifically, its abstract in DBpedia). Smoothing is done analogously to the column labels language model, but the components of the formula are computed differently:

$$P_{KB}(t|\theta_e) = \frac{tf(t,e) + \mu P(t|\theta)}{|e| + \mu} ,$$

where $tf(t,e)$ denotes the (raw) term frequency of t in the entity's description, $|e|$ is the length (number of terms) of that description, and $P(t|\theta)$ is a background language model (a maximum likelihood estimate from the descriptions of all entities in the KB).

To construct a term-based representation from the table corpus, we consider the captions of all tables that include entity e:

$$P_{TC}(t|e) = \frac{\#(t,e)}{\#(e)} ,$$

where $\#(t,e)$ denotes the number of tables that contain term t in the caption as well as entity e in the leftmost column. The denominator $\#(e)$ is the total number of tables containing e.[2]

5 POPULATING COLUMNS

In this section, we address the problem of column population using a two-step approach: we identify a set of candidate column heading labels (or *labels*, for short), and then subsequently rank them.

5.1 Candidate Selection

We use (i) the table caption, (ii) table entities, and (iii) seed column heading labels to search for similar tables. The searching method is the same as in §4.1, i.e., we use BM25 similarity using either of (i)–(iii) to get a ranking of tables from the table corpus. From these tables, we extract the column heading labels as candidates (excluding the seed column labels). When searching is done using the seed column labels as query, our method is equivalent to the FastJoin matcher [27] (which was also adopted in [12]).

5.2 Ranking Column Labels

We are interested in estimating the probability $P(l|E,c,L)$, given j seed labels, the table caption, and a set of entities from the rows.

[2]We also experimented with constructing a smoothed language model, similar to how it was done for the KB, but that gave inferior results.

5.2.1 Baseline Approach. Das Sarma et al. [8] consider the "benefits" of additional columns. The benefit of adding l to table T is estimated as follows:

$$P(l|L) = LB(L,l) = \frac{1}{|L|} \sum_{l_1 \in L} cs(l_1, l_2) , \quad (3)$$

where L denotes column labels and cs is the AcsDB [6] (*Attribute Correlation Statistics Database*) schema frequency statistics, which is given in Eq. (4). It is more effective to derive the benefit measure by considering the co-occurrence of pairs of labels, rather than the entire set of labels [8]. Eq. (4) determines the consistency of adding a new label l_2 to an existing label l_1:

$$cs(l_1, l_2) = P(l_2|l_1) = \frac{\#(l_1, l_2)}{\#(l_1)} , \quad (4)$$

where $\#(l_1, l_2)$ is number of tables containing both l_1 and l_2, and $\#(l_1)$ is the number of tables containing l_1.

5.2.2 Our Approach. Instead of estimating this probability directly, we use tables as a bridge. We search related tables sharing similar caption, labels, or entities with the seed table. Searching tables with only one aspect similarity is thought as a single method, e.g., searching tables with similar caption has the probability of $P(T|c)$. All these related tables are candidate tables acting as bridges. Each candidate table is weighted by considering its relevance with each candidate label, denoted as $P(l|T)$.

By applying the law of total probability, we get:

$$P(l|E,c,L) = \sum_T P(l|T) P(T|E,c,L) ,$$

where $P(l|T)$ is the label's likelihood given a candidate table (see §5.3), and $P(T|E,c,L)$ expresses that table's relevance (see §5.4).

5.3 Label Likelihood

Label likelihood, $P(l|T)$, may be seen as the importance of label l in a given table T. The simplest way of setting this probability is *uniformly* across the labels of the table:

$$P(l|T) = \begin{cases} 1, & \text{if } l \text{ appears in } T \\ 0, & \text{otherwise} . \end{cases}$$

5.4 Table Relevance Estimation

Table relevance expresses the degree of similarity between a candidate table and the seed table the user is working with. Tables with higher relevance are preferred. Specifically, we search for tables by considering the similarity of the set of entities, table caption, and column labels. The probability of a candidate table is factored as:

$$P(T|E,c,L) = \frac{P(T|E)P(T|c)P(T|L)}{P(T)^2} .$$

Notice that an independence assumption between E, c, and $L^{(j)}$ was made. Further, assuming that the prior probability of a table follows a uniform distribution, the denominator can be dropped. The components of this model are detailed below.

5.4.1 Entity Coverage. When selecting a candidate table, the coverage of the tables' entity set is a important factor [1, 8]. We compute the fraction of the seed table's entities covered by candidate table as:

$$P(T|E) = \frac{|T_E \cap E|}{|E|} .$$

We note that the same concept is used in [8], where it is referred to as *entity coverage*.

5.4.2 Caption Likelihood. Having similar captions is a strong indicator that two tables are likely to have similar contents. An effective way of calculating caption similarity is to use the seed table's caption as a query against a caption index of the table corpus. We can use any term-based retrieval model (like BM25 or language modeling) for measuring caption similarity:

$$P(T|c) \propto \text{sim}(T_c, c) .$$

5.4.3 Column Labels Likelihood. Finally, we estimate the column labels likelihood similar to Lehmberg et al. [12], who rank tables according to the number of overlapping labels:

$$P(T|L) = \frac{|T_L \cap L|}{|L|} .$$

6 EXPERIMENTAL DESIGN

We present the data sets we use in our experiments and our evaluation methodology. We develop an approach that simulates a user through the process of populating a seed table with data.

6.1 Data

We use the WikiTables corpus [3], which contains 1.6M tables extracted from Wikipedia. The knowledge base we use is DBpedia (version 2015-10). We restrict ourselves to entities which have an abstract (4.6M in total).

We preprocess the tables as follows. For each cell that contains a hyperlink we check if it points to an entity that is present in DBpedia. If yes, we use the DBpedia identifier of the linked entity as the cell's content (with redirects resolved); otherwise, we replace the link with the anchor text (i.e., treat it as a string).

6.2 Entity-Focused Tables

Recall that we defined an entity-focused table as one that contains only unique entities in its leftmost column (cf. §3). In addition to being an entity-focused table, we require that the table has at least 6 rows and at least additional 3 columns (excluding the entity column). We introduce these constraints so that we can simulate a real-world scenario with sufficient amount of content.

In Table 2, we report statistics based on what percentage of cells in the leftmost column contains entities. Let us note here that only those entities are recognized that have a corresponding Wikipedia article. Thus, the reported numbers should be treated as lower bound estimates. It is clear that many tables have an entity focus.

To be able to perform an automated evaluation without any human intervention, we apply the most strict conditions. Out of the tables that contain 100% unique entities in their leftmost column and have at least 6 rows and at least 4 columns (53 K in total), see Table 2, we randomly select 1000 tables as our validation set (used for parameter tuning) and another 1000 tables as our test set. We use a different random selection of validation/test tables for row and column population. The validation and test sets are excluded from the table corpus during training. It is important to note that we use all other tables from the corpus when computing statistics, and not only those that classify as entity-focused.

Table 2: Statistics of table corpus. Constraints mean having > 5 rows and > 3 columns.

leftmost column (X% are entities)	# tables total	# tables with constraints
Existing entity	726913	212923
60%	556644	139572
80%	483665	119166
100%	425236	78611
100% unique	376213	53354

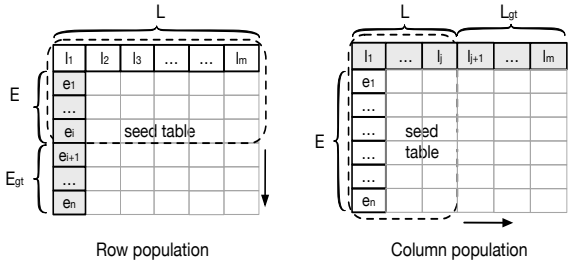

Figure 2: Illustration of our evaluation methodology. A part of an existing table is designated as seed table; the entities/column labels outside the seed table serve as ground truth. The arrows indicate the direction of the population.

6.3 Simulation Process

We evaluate row/column population by starting from an actual (complete) entity-focused table, with n content rows (with an entity in each) and m column headings. We simulate an user through the process of completing that table by starting with some seed rows/columns and iteratively adding one row/column at a time.

- For evaluating row population, we take entities from the first i rows ($i \in [1..5]$) as seed entities E, and use the entities from the remaining rows as ground truth, $\hat{E}$. We use all column heading labels.
- For evaluating column population, we take labels from the first j column ($j \in [1..3]$) as seed column labels L, and use the labels from the remaining columns as ground truth, $\hat{L}$. We use all entities from the table's rows.

See Figure 2 for an illustration. Notice that we are expanding in a single dimensions at a time; populating both rows and columns at the same time is left for future work.

6.4 Matching Column Labels

For the column population task, we are matching string labels (as opposed to unique identifiers). Let us consider `Date` as the ground truth column label. When the suggested labels are compared against this using strict string matching, then `date`, `Dates`, `date:`, etc. would not be accepted as correct, despite being semantically

identical. Therefore, we apply some simple normalization steps, on both the ranked and ground truth column labels, before comparing them using strict string matching. When multiple ranked labels are normalized to the same form, only the one with the highest score is retained.

6.5 Evaluation Metrics

Given that the relevance judgments are binary, we use Mean Average Precision (MAP) as our main evaluation metric. In addition, we also report on Mean Reciprocal Rank (MRR). We measure statistical significance using a two-tailed paired t-test. To avoid cluttering the discussion, we report significance testing only for our main metric.

7 EVALUATION OF ROW POPULATION

This section presents the evaluation of row population.

7.1 Candidate Selection

In §4.1, we have introduced four individual methods to select candidates: entity category (A1) and entity type (A2) from the knowledge base, and table caption (B) and table entities (C) from the table corpus. These methods involve a cut-off threshold parameter k; the top-k entities are considered as candidates for the subsequent ranking step. A larger k value typically implies higher recall. At the same time, each of the candidate entities will need to be scored, which is a computationally expensive operation. Therefore, we wish to find a setting that ensures high recall, while keeping the number of candidate entities manageably low (to ensure reasonable response time). We use the validation set to explore a range of k values: 2^6, 2^8, 2^{10}, and 2^{12}. For each method, we select the k value that produces the best recall and candidate entity number ratio.

The results are reported in the top block of Table 3. We observe that more seed entities we have, the better recall gets. This is expected behavior. Out of the two entity properties from the knowledge base, categories and types, categories performs far better. For types, even with $k = 4096$, the recall is still unsatisfactory. This is because many of the DBpedia entities have no ontology type information assigned to them. Moreover, ontology types are more general than categories and result in too many candidates. The best individual method is (C) table entities; it is the most effective (achieves the highest recall) and the most efficient (produces the lowest number of candidates) at the same time.

To further enhance performance, we combine the individual methods. However, we exclude type (A2) from this combination, because of its low performance. We find that all combinations improve over the single methods. This means that they capture complimentary aspects. Combining all three methods (A1+B+C) leads to the best overall performance. The last two lines of Table 3 show the performance of this combination (A1+B+C) using two different k values. We find that with a high k value (4096), we are able to achieve close to perfect recall. The number of candidates, however, is a magnitude larger than with a low k (256). Motivated by efficiency considerations, we decided not to pay this price and chose to use $k = 256$, which still gives us very high recall.

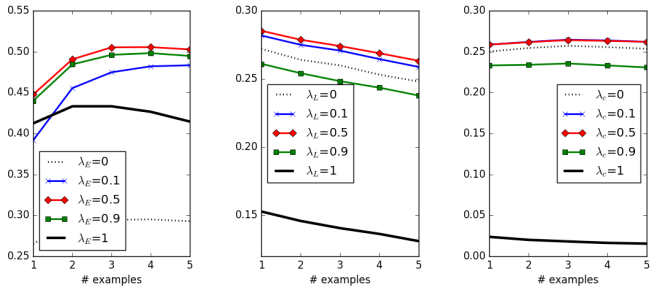

Figure 3: Effect of varying the interpolation parameters for $P(e|E)$ (Left), $P(L|e)$ (Middle), and $P(c|e)$ (Right). The plots show MAP scores measured on the validation set.

7.2 Entity Ranking

Our entity ranking model is comprised of three components: entity similarity ($P(e|E)$), column labels likelihood ($P(L|e)$), and caption likelihood ($P(c|e)$). Each of these methods involve an interpolation parameter (λ_E, λ_L, and λ_c, respectively). We train these parameters on the validation set, by performing a sweep in 0.1 steps over the [0..1] range. The effect of varying the parameter values is shown in Figure 3. It can be seen that the value 0.5 provides the best setting everywhere. We also found that there is very little difference in terms of performance when λ is in the 0.3..0.7 range (hence the choice of showing the 0.1 and 0.9 values on the plots).

We start by discussing the performance of individual components, reported in the top block of Table 4. The two-component entity similarity model combines estimated based on the knowledge base and the table corpus (cf. Eq. (2)). For the former, we compare three alternatives: using relations of entities, as in [4] (A1), and two similarity methods based on outgoing links of entities: WLM (A2), and Jaccard similarity (A3). Out of the three methods, (A1) Relations has the best performance. However, (A3) has only marginally lower retrieval performance, while being computationally much more efficient. Therefore, we choose (A3), when it comes to combining it with the other elements of the entity ranking model. Compared to entity similarity ($P(e|E)$), the other two components (B and C) have much lower performance. The differences (A3) vs. (B) and (A3) vs. (C) are highly significant ($p < 10^{-5}$). This means that the knowledge base contributes more.

Next, we combine the individual components to further enhance performance. The middle block of Table 4 reports results when two components are used. We find that these combinations improve significantly over the individual methods in all cases ($p < 10^{-5}$). It is interesting to note that while (C) caption likelihood outperforms (B) column labels likelihood in the individual comparison (significantly so for #1..#3 seed entities, $p < 0.001$), the two perform on a par when combined with (A3) entity similarity.

As expected, using all three component (A3 & B & C) results in the best performance. The differences between this vs. (A3 & C) and vs. (B & C) are significant for any number of seed entities ($p < 0.001$); regarding (A3 & B & C) vs. (A3 & B), the differences are significant only for seed entities #1 and #5 ($p < 0.05$). This means that combining information from the knowledge base with column

Table 3: Candidate selection performance for the row population task on the validation set. #cand refers to the number of candidate entities. Highest recall values are typeset in boldface.

Method	#Seed entities ($\|E\|$)									
	1		2		3		4		5	
	Recall	#cand	Recall	#cand	Recall	#cand	Recall	#cand	Recall	#cand
(A1) Categories (k=256)	0.6470	1721	0.6985	2772	0.7282	3678	0.7476	4507	0.7604	5224
(A2) Types (k=4096)	0.0553	7703	0.0577	8047	0.0585	8225	0.0605	8419	0.0600	8551
(B) Table caption (k=256)	0.3966	987	0.3961	987	0.3945	987	0.3938	987	0.3929	987
(C) Table entities (k=256)	0.6643	312	0.7212	458	0.7435	589	0.7564	689	0.7639	759
(B) & (C) (k=256)	0.7090	1250	0.7464	1383	0.7626	1505	0.7732	1599	0.7788	1664
(A1) & (B) (k=256)	0.7642	2671	0.7969	3711	0.8157	4610	0.8305	5434	0.8405	6145
(A1) & (C) (k=256)	0.8434	1962	0.8885	3118	0.9038	4117	0.9196	5014	0.9285	5773
(A1) & (B) & (C) (k=256)	0.8662	2880	0.8997	4018	0.9154	5005	0.9255	5894	0.9329	6645
(A1) & (B) & (C) (k=4096)	**0.9576**	28733	**0.9718**	40171	**0.9787**	49478	**0.9811**	58021	**0.9821**	65204

labels from the table corpus yields significant benefits; considering the captions of tables on top of that leads to little additional gain.

For baseline comparison, we employ the method by [4], which combines text-based and structure-based similarity. Note that we used only the structure-based part of their method earlier, as (A1); here, we use their approach in its entirety. It requires a keyword query, which we set to be the table caption. We find that our methods substantially and significantly ($p < 10^{-5}$) outperforms this baseline; see the bottom two rows in Table 4.

One final observation is that performance climbs when moving from a single to two and three seed entities; after that, however, it plateaus. This behavior is consistent across all methods, including the baseline. The phenomena is known from prior work [4, 16, 20].

7.3 Analysis

Now that we have presented our overall results, we perform further examination on the level of individual tables. Figure 4 shows the average precision (AP) scores for the 1000 test tables, ordered by decreasing score. Statistically, there are 285 tables having $AP = 1$, 193 tables having $0.4 < AP < 0.6$, and 42 tables having $AP = 0$. To understand the reasons behind this, we check the recall of the candidate selection step for these three categories; see Figure 5. In this figure, we can observe that higher recall generally leads to better AP. Delving deeper, we compute the average number of tables containing at least one ground truth entity, for each of the three groups. When $AP = 0$, the number is 18, for $0.4 < AP < 0.6$ it is 79, and for $AP = 1$ it is 127. It appears that we could provide excellent suggestions, when there were enough similar tables to the seed table in the table corpus. However, for tables that are "too unique," we would need alternative methods for suggestions.

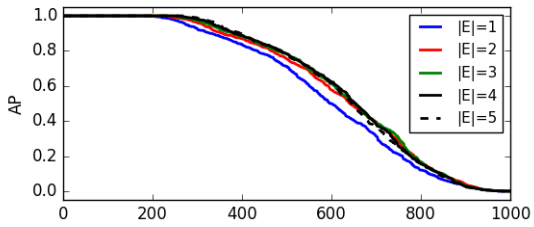

Figure 4: Performance of individual tables, ordered by decreasing Average Precision, for row population.

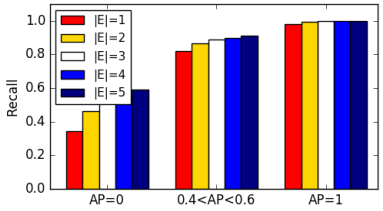

Figure 5: Recall of candidate selection against entity ranking performance, for row population.

8 EVALUATION OF COLUMN POPULATION

This section presents the evaluation of column population. This task relies only on the table corpus; the data set is exactly the same as for row population, see §6.1.

8.1 Candidate Selection

In §5.1, we have introduced three individual methods to select candidates: table caption (A), column heading labels (B) and table entities (C). Method (B) actually corresponds to the FastJoin matcher in [27]. These methods also involve a cut-off threshold parameter k, for the same reasons we already discussed in §7.1. The results are reported in the top block of Table 5. We observe that the more seed labels we have the better recall gets when using labels. We also explore combinations of pairs of methods as well as using all three. We find that all combinations improve over the single methods, and that combining all three methods leads to the best overall performance. Our selected method is the second to last in Table 5, motivated by efficiency considerations; for comparison, we also show the performance for $k = 4096$.

8.2 Column Label Ranking

Our column label ranking model is comprised of two components: table relevance and label likelihood. For estimating candidate table relevance, we have three individual methods, using table caption

Table 4: Entity ranking performance on the test set.

Method	#Seed entities ($\|E\|$) 1		2		3		4		5	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
(A1) $P(e\|E)$ Relations ($\lambda = 0.5$)	0.4962	0.6857	0.5469	0.7297	0.5687	0.7415	0.5734	0.7294	0.5693	0.7274
(A2) $P(e\|E)$ WLM ($\lambda = 0.5$)	0.4674	0.6246	0.5154	0.6901	0.5293	0.6930	0.5331	0.6861	0.5258	0.6789
(A3) $P(e\|E)$ Jaccard ($\lambda = 0.5$)	0.4905	0.6731	0.5427	0.7086	0.5617	0.7270	0.5662	0.7098	0.5609	0.7058
(B) $P(L\|e)$	0.2857	0.3558	0.2878	0.3518	0.2717	0.3463	0.2651	0.3365	0.2585	0.3336
(C) $P(c\|e)$	0.2348	0.2656	0.2366	0.2676	0.2371	0.2656	0.2350	0.2614	0.2343	0.2602
(A3) & (B)	0.5726	0.7593	0.6108	**0.8055**	0.6189	**0.7879**	0.6182	0.7755	0.6108	**0.7689**
(A3) & (C)	0.5743	0.7467	0.6108	0.7749	0.6221	0.7746	0.6211	0.7668	0.6156	0.7447
(B) & (C)	0.3677	0.4521	0.3715	0.4508	0.3712	0.4455	0.3688	0.4408	0.3667	0.4378
(A3) & (B) & (C)	**0.5922**	**0.7729**	**0.6260**	0.8000	**0.6339**	0.7849	**0.6348**	**0.7800**	**0.6310**	0.7630
Baseline [4]	0.3076	0.4967	0.3273	0.5156	0.3404	0.5326	0.3428	0.5290	0.3406	0.5202

Table 5: Candidate selection performance for the column population task on the validation set.

Method	#Seed column labels ($\|L\|$) 1		2		3	
	Recall	#cand	Recall	#cand	Recall	#cand
(A) Table caption (k=256)	0.7177	232	0.7115	232	0.7135	231
(B) Column labels (k=256)	0.2145	115	0.5247	235	0.7014	357
(C) Table entities (k=64)	0.7617	157	0.7544	156	0.7505	155
(A) (k=256) & (B) (k=256) & (C) (k=64)	0.8799	467	0.8961	572	0.9040	682
(A) (k=4096) & (B) (k=4096) & (C) (k=4096)	**0.9211**	2614	**0.9292**	3309	**0.9351**	3978

(A), column labels (B), and table entities (C). All methods use the same estimation of label likelihood (cf. §5.3).

We start by discussing the performance of individual methods, which is reported in the top block of Table 6. Of the three, method (C) outperforms the other two, and does significantly so ($p < 10^{-5}$). Looking at the tendency of MAP, the increasing number of seed column labels only contributes to method (B). When combining two of the methods, all combinations improve significantly over the individual methods ($p < 10^{-5}$). Out of the three, (B) & (C) performs best in terms of both *MAP* and *MRR*. In the end, putting together all three individual methods delivers the best results. Also, this combination (A & B & C) improves significantly over the combination of any two of the methods ($p < 10^{-5}$).

For baseline comparison, we employ the method by Das Sarma et al. [8]. They consider the "benefits" of adding additional columns, which expressed in Eq. (3). We find that our three-component method substantially and significantly ($p < 10^{-5}$) outperforms this baseline. It should be noted that the baseline in [8] uses our candidate selection method to make it comparable; this actually performs better than their original approach.

8.3 Analysis

Figure 6 plots the performance of individual (test) tables, in decreasing order of average precision score. We find that there are 427 tables having $AP = 1$, 122 tables having $0.4 < AP < 0.6$, and 186 tables having $AP = 0$. We examine these three table groups, based on performance, in terms of their corresponding recall values from the candidate selection step. Figure 7 shows these values (averaged over all tables that fall in the given performance group). Looking at the number of tables containing at least one ground truth column heading label, it is 204 for $AP = 0$, 403 for $0.4 < AP < 0.6$, and 1114 for $AP = 1$. We can draw similar conclusions here as we did for entity ranking.

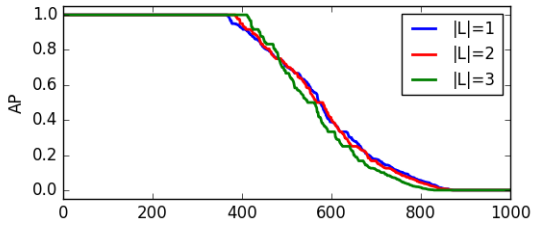

Figure 6: Performance of individual tables, ordered by decreasing Average Precision, for column population.

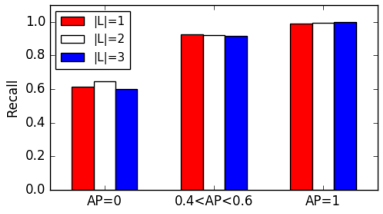

Figure 7: Recall of candidate selection against column label ranking performance, for column population.

Table 6: Column label ranking performance on the test set.

Method	#Seed column labels ($\|L\|$)					
	1		2		3	
	MAP	MRR	MAP	MRR	MAP	MRR
(A) Table caption	0.2584	0.3496	0.2404	0.2927	0.2161	0.2356
(B) Column labels	0.2463	0.3676	0.3145	0.4276	0.3528	0.4246
(C) Table entities	0.3878	0.4544	03714	0.4187	0.3475	0.3732
(A) & (B)	0.4824	0.5896	0.4929	0.5837	0.4826	0.5351
(A) & (C)	0.5032	0.5941	0.4909	0.5601	0.4724	0.5132
(B) & (C)	0.5060	0.5954	0.5410	0.6178	0.5323	0.5802
(A) & (B) & (C)	**0.5863**	**0.6854**	**0.5847**	**0.6690**	**0.5696**	**0.6201**
Baseline [8]	0.4413	0.5473	0.4640	0.5535	0.4535	0.5079

9 CONCLUSION

In this paper, we have introduced the idea of a smart table assistant and have taken the first steps towards its realization. Specifically, we have concentrated on tables with an entity focus, and investigated the tasks of row population and column population. We have devised methods for each task and showed experimentally how the different components all contribute to overall performance. For evaluation, we have developed a process that simulates a user through her work of populating a table with data. Our overall results are very promising and substantially outperform existing baselines.

In future work, we plan to extend the capabilities of our assistant to be able to populate data cells as well with values. Further along the road, we also wish to relax our requirement regarding the entity focus, and make our methods applicable to arbitrary tables.

REFERENCES

[1] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. 2015. Towards a Hybrid Imputation Approach Using Web Tables.. In *Proc. of BDC '15*. 21–30.

[2] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2013. Methods for Exploring and Mining Tables on Wikipedia. In *Proc. of IDEA '13*. 18–26.

[3] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proc. of ISWC 2015*. 425–441.

[4] Marc Bron, Krisztian Balog, and Maarten de Rijke. 2013. Example Based Entity Search in the Web of Data. In *Proc. of ECIR'13*. 392–403.

[5] Michael J. Cafarella, Alon Halevy, and Jayant Madhavan. 2011. Structured Data on the Web. *Commun. ACM* 54 (2011), 72–79.

[6] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: Exploring the Power of Tables on the Web. *Proc. VLDB Endow.* 1 (2008), 538–549.

[7] Eric Crestan and Patrick Pantel. 2011. Web-scale Table Census and Classification. In *Proc. of WSDM '11*. 545–554.

[8] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding Related Tables. In *Proc. of SIGMOD '12*. 817–828.

[9] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proc. of KDD '14*. 601–610.

[10] Yeye He and Dong Xin. 2011. SEISA: set expansion by iterative similarity aggregation. In *Proc. of WWW '11*. 427–436.

[11] Yusra Ibrahim, Mirek Riedewald, and Gerhard Weikum. 2016. Making Sense of Entities and Quantities in Web Tables. In *Proc. of CIKM '16*. 1703–1712.

[12] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. 2015. The Mannheim Search Join Engine. *Web Semant.* 35 (2015), 159–166.

[13] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. of VLDB Endow.* 3 (2010), 1338–1347.

[14] Aman Madaan, Ashish Mittal, Mausam, Ganesh Ramakrishnan, and Sunita Sarawagi. 2016. Numerical Relation Extraction with Minimal Supervision. In *Proc. of AAAI'16*. 2764–2771.

[15] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Y. Halevy. 2009. Harnessing the Deep Web: Present and Future. *CoRR* abs/0909.1785 (2009).

[16] Steffen Metzger, Ralf Schenkel, and Marcin Sydow. 2014. Aspect-Based Similar Entity Search in Semantic Knowledge Graphs with Diversity-Awareness and Relaxation. In *Proc. of WI-IAT '14*. 60–69.

[17] David Milne and Ian H. Witten. 2008. An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. In *In Proc. of AAAI 2008*.

[18] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. 2014. Using Linked Data to Mine RDF from Wikipedia's Tables. In *Proc. of WSDM '14*. 533–542.

[19] Varish Mulwad, Tim Finin, and Anupam Joshi. 2013. Semantic Message Passing for Generating Linked Data from Tables.. In *Proc. of ISWC 2013*. 363–378.

[20] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. 2009. Web-scale Distributional Similarity and Entity Set Expansion. In *Proc. of EMNLP '09*. 938–947.

[21] S. Roy, T. Vieira, and D. Roth. 2015. Reasoning about Quantities in Natural Language. *Transactions of the Association for Computational Linguistics* 3 (2015).

[22] Sunita Sarawagi and Soumen Chakrabarti. 2014. Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models. In *Proc. of KDD '14*. 711–720.

[23] Yoones A. Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. 2014. Knowledge Base Augmentation using Tabular Data. In *Proceedings of WWW'14)*.

[24] Thibault Sellam and Omar Alonso. 2015. Raimond: Quantitative Data Extraction from Twitter to Describe Events. In *Proc. of ICWE 2015*. 251–268.

[25] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables on the Web. *Proc. VLDB Endow.* 4 (2011), 528–538.

[26] Chi Wang, Kaushik Chakrabarti, Yeye He, Kris Ganjam, Zhimin Chen, and Philip A. Bernstein. 2015. Concept Expansion Using Web Tables. In *Proc. of WWW '15*. 1198–1208.

[27] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2014. Extending String Similarity Join to Tolerant Fuzzy Token Matching. *ACM Trans. Database Syst.* 39, 1 (2014), 1–45.

[28] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. 2012. Understanding Tables on the Web. In *Proceedings of ER'12*. 141–155.

[29] Richard C. Wang and William W. Cohen. 2008. Iterative Set Expansion of Named Entities Using the Web. In *Proc. of ICDM '08*. 1091–1096.

[30] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proc. of SIGMOD '12*. 97–108.

[31] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural Enquirer: Learning to Query Tables in Natural Language. In *Proc. of IJCAI '16*. 2308–2314.

[32] Meihui Zhang and Kaushik Chakrabarti. 2013. InfoGather+: Semantic Matching and Annotation of Numeric and Time-varying Attributes in Web Tables. In *Proc. of SIGMOD '13*. 145–156.

[33] Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. 2013. Towards Disambiguating Web Tables. In *Proc. of ISWC-PD'13*. 205–208.