Auto-completion for Data Cells in Relational Tables

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

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

ABSTRACT

We address the task of auto-completing data cells in relational tables. Such tables describe entities (in rows) with their attributes (in columns). We present the CellAutoComplete framework to tackle several novel aspects of this problem, including: (i) enabling a cell to have multiple, possibly conflicting values, (ii) supplementing the predicted values with supporting evidence, (iii) combining evidence from multiple sources, and (iv) handling the case where a cell should be left empty. Our framework makes use of a large table corpus and a knowledge base as data sources, and consists of preprocessing, candidate value finding, and value ranking components. Using a purpose-built test collection, we show that our approach is 40% more effective than the best baseline.

CCS CONCEPTS

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

KEYWORDS

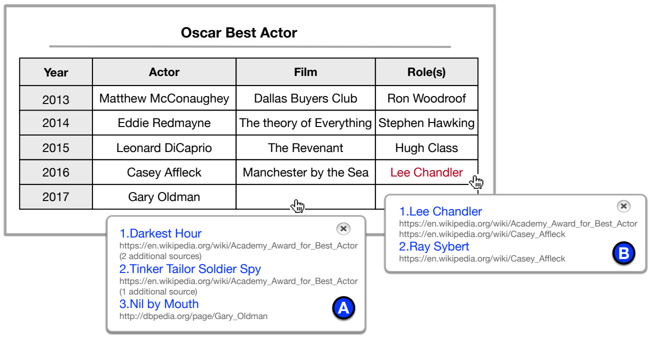
Structured data search; data augmentation; table cell value finding; table auto-completion; table matching

ACM Reference Format:
Shuo Zhang and Krisztian Balog. 2019. Auto-completion for Data Cells in Relational Tables. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357932

1 INTRODUCTION

Tables are a frequently used tool for collecting information about entities of interest. Relational tables (also referred to as entity-attribute tables [20]) are a particular type of table utilized for that purpose, where rows correspond to entities and columns corresponds to attributes of those entities. There is a growing body of research on assisting users in the labor-intensive process of table creation by helping them to augment tables with data [1, 20, 23], retrieve existing tables [1, 20, 24], and even automatically generate entire tables [25]. This paper falls in the category of *data augmentation* (also referred to as *data imputation* [1]), which is concerned with extending a given input table with more data. Specifically, we propose to equip spreadsheet applications with a kind of *auto-complete* functionality, where, upon clicking on a cell, the user is presented with a list of suggestions of possible values for that cell; see Fig. 1. This task has in fact been addressed in [23] for row and column heading cells of relational tables. Our goal in this paper is to provide a similar auto-complete feature for *value cells*. This is a considerably more difficult problem that poses a range of unique challenges.

CIKM '19, November 3–7, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357932



Oscar Best Actor

Year	Actor	Film	Role(s)
2013	Matthew McConaughey	Dallas Buyers Club	Ron Woodroof
2014	Eddie Redmayne	The theory of Everything	Stephen Hawking
2015	Leonard DiCaprio	The Revenant	Hugh Class
2016	Casey Affleck	Manchester by the Sea	Lee Chandler
2017	Gary Oldman		

1.Darkest Hour
https://en.wikipedia.org/wiki/Academy_Award_for_Best_Actor
(2 additional sources)
2.Tinker Tailor Soldier Spy
https://en.wikipedia.org/wiki/Academy_Award_for_Best_Actor
(1 additional source)
3.Nil by Mouth
http://dbpedia.org/page/Gary_Oldman

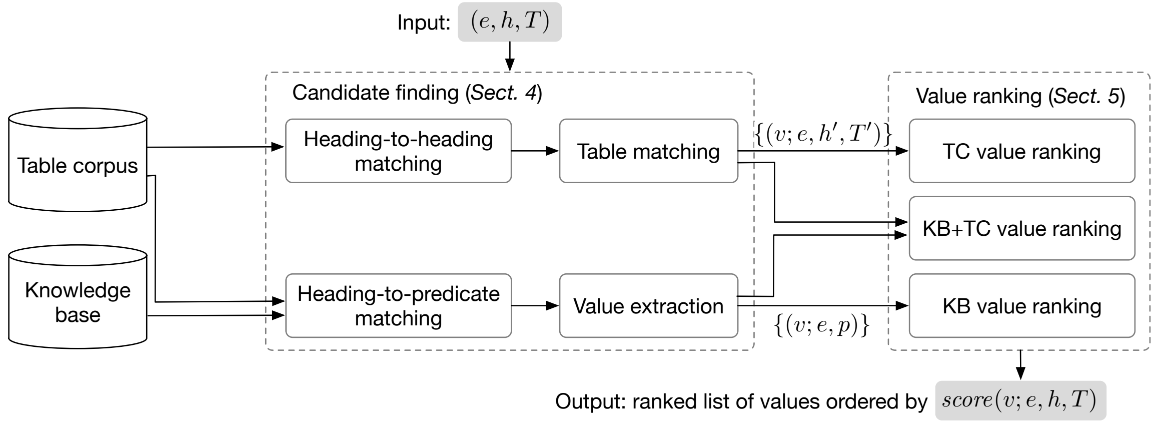
1.Lee Chandler
https://en.wikipedia.org/wiki/Academy_Award_for_Best_Actor
https://en.wikipedia.org/wiki/Casey_Affleck
2.Ray Sybert
https://en.wikipedia.org/wiki/Casey_Affleck

Figure 1: Envisioned user interface. By clicking on a table cell, the user receives a ranked list of suggested values along with supporting sources. Case (A) is to find the value for an empty cell. Case (B) is to check/verify an existing value.

First, it is paramount that each value that is offered to the user in the list of suggestions for a given cell has *supporting evidence*. That is, the user can verify that value by tracing it back to its source of origin. Intuitively, the more support a given value has the higher it should be ranked on the list of suggestions. This renders model-based approaches, which train a machine learning model to guess the missing values [1], unsuitable for this scenario. Second, it is important to recognize when a cell should in fact be left empty, and to not deceive the user with nonsense or misleading suggestions.

Addressing these challenges requires a fundamentally different approach from prior work. Our CellAutoComplete framework consists of two main steps. First, we identify candidate values from a table corpus and from a knowledge base. A key component in this process is mapping the target attribute (e.g., "venue") to (i) column heading labels in the table corpus that have the same meaning (e.g., "stadium") and (ii) to predicates in a knowledge base (e.g., `<dbo:ground>`). Second, we combine numerous signals in a learning-to-rank (LTR) framework to generate a ranking of the candidate values. A particularly novel idea, that is captured by a specific group of features, is to consider the semantic similarity of each candidate table, which mentions the target entity and attribute, to the input table. In order to deal with cells that should be left empty, we introduce a special designated `Empty` value. This allows us to quantify our belief that a given cell should be left empty. Our experimental results show that our LTR approach outperforms the best single-source baseline by about 40% in terms of NDCG@10.

Our contributions can be summarized as follows:



Input: (e, h, T)
Candidate finding (*Sect. 4*)
Table corpus
Knowledge base
Heading-to-heading matching
Table matching
Heading-to-predicate matching
Value extraction
$\{(v; e, h', T')\}$
$\{(v; e, p)\}$
Value ranking (*Sect. 5*)
TC value ranking
KB+TC value ranking
KB value ranking
Output: ranked list of values ordered by $score(v; e, h, T)$

Figure 2: Overview of the CellAutoComplete framework.

- We present the CellAutoComplete framework for finding cell values in relational tables (Sect. 3), which consists of candidate value finding (Sect. 4) and value ranking (Sect. 5) components. Specific novel technical contributions include the heading-to-heading and heading-to-predicate matching components (§4.1.1 and §4.2.1) as well as the features designed for combining evidence from multiple sources and for predicting empty values (Sect. 5.3).
- We develop a purpose-built test collection based on Wikipedia tables, comprising 35k manually labeled cell-value pairs for 1000 cells (Sect. 6), and perform an extensive experimental evaluation (Sect. 7). Our experiments show that our LTR approach substantially outperforms existing data augmentation techniques.

All resources developed in this study are made publicly available at https://github.com/iai-group/cikm2019-table.

2 RELATED WORK

Table augmentation is the task of extending an input table with more data [1, 3, 5, 6, 12, 18, 20, 22, 23]. This task may be performed on different levels: (i) simply finding related tables, (ii) augmenting individual cells, such as populating heading rows and columns with additional entities and column labels or finding values for data cells, and (iii) adding entire rows/columns.

To find related tables that are potentially useful, Das Sarma et al. [6] search for *entity complement* tables that are semantically related to entities in the input table, and *schema complement* tables for augmenting table schema. Aiming for more targeted (cell-level) augmentation, Zhang and Balog [23] propose intelligent assistance functionalities for populating row and column headings of relational tables. Specifically, they develop probabilistic models for ranking a list of suggestions, that is, entities to be added as headings of new rows and labels to be added as headings of new columns. Both subtasks rely on a knowledge base and on similar tables retrieved from a table corpus as sources of suggestions. A recent study utilizes Word2vec to train embeddings for these tasks [7]. Yakout et al. [20] present the InfoGather system that performs table augmentation in three flavors: *augmentation by example*, *schema auto-complete* and *augmenting attributes*, designed for augmenting table entities, heading labels, and table cell values, respectively. The former two tasks are the same as those in [23]. The last of these, augmenting attributes, is particularly relevant for the current paper. The approach taken in [20] is to first search for similar tables, then to fuse the corresponding cell values from the matching tables. Yakout et al. [20] match entities by value overlap. To match column labels, they propose a holistic approach of utilizing additional information including similarities based on context, attribute names, and column values, to overcome the shortcomings of traditional schema matching and instance level features. We use InfoGather as a baseline in our experiments. With a similar mission, Zhang and Chakrabarti [22] extend the system as InfoGather+, focusing on numerical and time-varying attributes augmentation.

The above tasks are extending tables cell by cell. There is also a body of work on joining (entire) rows/columns from existing tables. Bhagavatula et al. [3] propose the *relevant join* task, which returns a ranked list of column triplets for a given input table. The first two elements are the input column and the matched column, and the third element quantifies the correlation between them. The remaining columns that co-exist with the matched columns are taken as candidates. They employ a linear model to rank the candidate columns, which could be joined to the input table as an additional column. Lehmberg et al. [12] design a join search engine, which searches related tables based on table column headings, then applies a series of left outer joins by taking columns from returned tables and adding them to the input table. Focusing on table cell values, Ahmadov et al. [1] propose a combined method, depending on the features of missing values, so as to look missing values from web data, predict them utilizing machine learning models, or combine the above two to find the most likely values.

While it is not a table augmentation approach, the work by Zhang and Balog [25] is also highly relevant to our task. The authors propose to answer entity-oriented queries by generating relational tables as answers "on the fly." They first determine the entities (heading rows) and their attributes (heading columns), then find the values of the corresponding cells. They build a value catalog based on matching values found in Wikipedia tables and in DBpedia, giving priority to the latter, from which they can then fetch cell values. We use their method as a baseline in our experiments.

Our work is also related to *truth/fact finding* research. Yin et al. [21] build a fact lookup engine, FACTOR, based on web tables. FACTOR can answer fact lookup queries, which are about a certain attribute value of one entity, e.g., the birth date of Taylor Swift.

Specifically, they extract entity-attribute-value triples from tables on the Web, aggregate and clean them, and store them in a database. For a given query, FACTOR generates every possible combination of entity and attribute, and retrieves a set of values, which are ranked considering entity similarity, attribute similarity, URL format, and value types. Ernst et al. [10] present HighLife to harvest higher-arity facts from texts, in order to capture more complete and deeper knowledge about events or multi-entity relationships. The method is distantly supervised by seed facts, which are facts that have been verified. Facts found in tables can be used for question answering. For example, Sun et al. [17] propose a deep matching model for matching question and table cells. The matched table cells are taken as the answers. These approaches take free text queries as input, while our task considers a table as input. This makes the nature of the task quite different, and renders those approaches unsuitable for us as baselines.

Summary of differences. There are several aspects that distinguish our work from relevant existing approaches. First, prior work has limited the task of value finding to that of identifying a single best value. Conflicting data, however, can coexist [19]. Casting value finding as a ranking problem provides a mechanism to deal with this plurality. Second, most prior work relies on a single source, a table corpus, for finding missing values [1, 20, 22]. We also use a knowledge base, in addition, in case there are no relevant tables for the target entity and attribute on the Web. The combination of a table corpus and a knowledge base has already been exploited in [25]. There, however, only the single best source is used. Instead, we combine evidence from the two sources. Third, existing work either does not consider the data types of table columns [1, 20, 25] or is limited to numerical values [1, 22]. We consider multiple value types, including non-numerical ones (entities and strings).

3 PROBLEM STATEMENT AND OVERVIEW

We address the task of automatically finding the values of cells in relational tables. A table is said to be relational if it describes a set of entities in its core column (typically, the leftmost column) and the attributes of those entities in additional columns. We shall assume that entities in the core column of each table have been identified and linked to a knowledge base. These annotations may be supplied manually (e.g., tables in Wikipedia) or can be obtained automatically [9, 16]. The additional (attribute) columns are identified by their heading labels.

Formally, given an input table T, we seek to find the value of the cell that is identified by the row with entity e (in the core column) and the column with heading label h. The output is a ranked list of values v, where the ranking of values is defined by a scoring function $score(v; e, h, T)$.

Our approach, shown in Fig. 2, has two main components. First, we identify candidate values from two sources, a table corpus and a knowledge base. Second, these values are ranked based on their likelihood of being correct, and the top-k ranked values are presented to the user as auto-complete suggestions. Note that it is a design decision for us to keep the user in the loop and let her make a judgment call on the appropriateness of a suggestion by considering the supporting evidence.

The two main components of our CellAutoComplete framework are described in the following two sections.

4 CANDIDATE VALUE FINDING

In this section, we address the problem of identifying candidate values for a given target cell in table T, identified by the target entity e and target heading label h. Candidate value finding is a crucial step as the recall of the end-to-end task critically depends on it. We gather candidate values from two sources: a table corpus (Sect. 4.1) and a knowledge base (Sect. 4.2). Novel contributions in this part include the heading-to-heading and heading-to-predicate matchings, and the TMatch table matching approach.

4.1 Table Corpus

Our goal is to locate tables from the corpus that contain the target entity and attribute (heading label) pair. We assume a setting where entities in the core table columns have been linked to a knowledge base (cf. Sect. 6.1). With that, it is easy to find the tables that contain the target entity (with high confidence). The matching of heading labels, however, is not that straightforward: the same meaning may be expressed using different labels (e.g., “established” vs. “founded”), while the same label can mean different things depending on the table’s context (e.g., the column label “played” may refer, among others, to the number of games played, to the date of a game, or the name of the opponent). Therefore, we need to perform a matching between heading labels (§4.1.1). Naively considering all candidate tables that mention the entity and heading is not sufficient; additionally, we should also take into account their semantic similarity to the input table, referred to as the problem of *table matching* (§4.1.2).

4.1.1 Heading-to-Heading Matching. For a given heading label h, we wish to identify additional heading labels that have the same meaning (i.e., refer to the same entity attribute). This is closely related to the problem of schema matching [8]. The main idea is that if two tables T_a and T_b contain the same value v for a given entity e in columns with headings h_a and h_b, respectively, then h_a and h_b *might* mean the same. Sharing a value does not always mean the equivalence of heading labels. Nevertheless, the intuition is that the more often it happens, the more likely it is that h_a and h_b refer to the same entity attribute. We capture this intuition in the following formula:

$$P(h'|h) = \frac{n(h', h)}{\sum_{h''} n(h'', h)} , \qquad (1)$$

where $n(h', h)$ is the number of table pairs in the table corpus that contain the same value for a given entity in columns h' and h, respectively.

4.1.2 Table Matching. All tables in the table corpus that contain (i) the target entity e and (ii) the target heading h or any related heading label h' ($n(h', h) > 0$), are considered as candidates. As we explained above, not all these tables are actually good candidates. Therefore, we estimate the semantic similarity between the input table T and a candidate table T', $score(T, T')$. This table matching score later will be utilized as a confidence estimate in a subsequent value ranking step (in Sect. 5). We present two feature-based learning methods for table matching. We start with a state-of-the-art approach, InfoGather. Then, we introduce TMatch, which extends InfoGather with a rich set of features from the literature.

Table 1: Overview of table ranking features used in TMatch.

Group / Feature	Source
Table features	
Number of rows in the table	[3, 5]
Number of columns in the table	[3, 5]
Number of empty table cells	[3, 5]
Table caption IDF	[15]
Table page title IDF	[15]
Number of in-links to the page embedding the table	[3]
Number of out-links from the page embedding the table	[3]
Number of page views	[3]
Inverse of number of tables on the page	[3]
Ratio of table size to page size	[3]
Matching features	
InfoGather page title IDF similarity score	[20]
InfoGather heading-to-heading similarity	[20]
InfoGather column-to-column similarity	[20]
InfoGather table-to-table similarity	[20]
MSJE heading matching score	[12]
Nguyen et al. [14] heading similarity	[14]
Nguyen et al. [14] table data similarity	[14]
Schema complement schema benefit score	[6]
Schema complement entity overlap score	[6]
Entity complement entity relatedness score	[6]

InfoGather. InfoGather [20] measures element-wise similarities across four table elements (table data, column values, page title, and heading labels), and combines them in a linear fashion:

$$score(T, T') = \sum_x w_x \times sim(T_x, T'_x) ,$$

where x refers to a given table element. Each table element T_x is expressed as a term vector. Element-wise similarity $sim()$ is computed using the cosine similarity between the respective term vectors of the input and candidate tables.

TMatch. We extend the four element-wise matching scores of InfoGather with a number of additional matching, which are summarized in Table 1. We use Random Forests regressor as our machine-learned model. We distinguish between two main groups of features. The first group of features (top block in Table 1) aim to characterize an individual table and are associated with its quality and importance. These features are computed for both the input and candidate tables. The second group of features (bottom block in Table 1) measures the degree of matching between the input and candidate tables. In the interest of space, we present a high-level description of these features and refer to the original publications for details.

- The Mannheim Search Join Engine (MSJE) [12] measures the similarity between the headings of two tables by creating an edit distance graph between the input and candidate tables' heading terms. Then, the *maximum weighted bipartite matching score* is computed on this graph's adjacency matrix.
- Nguyen et al. [14] consider the table headings and table data for matching. Specifically, heading similarity is computed by solving the *maximum weighted bipartite sub-graph problem* [2]. Data similarity is measured by representing each table column as a binary term vector, and then taking the cosine similarity between the most similar column pairs.

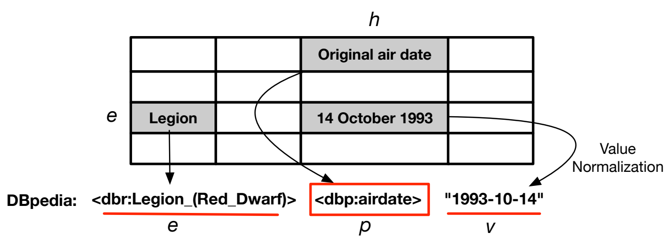


Figure 3: Illustration of table heading label to KB predicate matching. Notice that value normalization is also involved.

Table 2: Examples of heading label and predicate matches.

Heading label (h)	Predicate (p)	$n(h,p)$
Director	dbp:director	38587
	dbp:writer	2348
Location	dbp:city	10772
	dbp:location	9170
Country	dbp:birthPlace	9077
	dbp:country	3546

- Das Sarma et al. [6] compute the matching score by aggregating the benefits of adding additional headings columns and entities from the candidate table to the input table.

4.2 Knowledge Base

Next, we discuss how to utilize a knowledge base for gathering candidate values. By definition, the columns in relational tables correspond to entity attributes. The main challenge to be addressed here is how to map the column heading label to the appropriate KB predicate (§4.2.1). After that, candidate values can easily be fetched from the KB (§4.2.2).

4.2.1 Heading-to-Predicate Matching. Both table heading labels and knowledge base predicates represent entity attributes, but these are often expressed differently, making string matching insufficient. Similarly to how it was done in heading-to-heading matching, we capitalize on the observation that if entity e has value v for predicate p in the KB, and the same entity has value v in the heading column h of many tables, then p and h are likely to mean the same (more precisely, h is a string label that corresponds to the semantic relation p). This idea is illustrated in Fig. 3. The similarity between heading label (h) and predicate (p) is computed according to the conditional probability $P(p|h)$:

$$P(p|h) = \frac{n(h,p)}{\sum_{p'} n(h,p')} , \quad (2)$$

where $n(h,p)$ denotes the times of h and p indicate the same value in the corpus. Table 2 lists some examples.

4.2.2 Value Extraction. Given a table heading, all matching predicates (i.e., where $n(h,p) > 0$) are considered. Then, for each of these predicates p, the object values associated with the subject e and predicate p in the knowledge base are considered as candidate values (i.e., all the subjects of SPO triples matching the pattern $\langle e, p, ?\rangle$).

5 VALUE RANKING

In this section we describe methods for ranking the candidate cell values that were identified in the previous step. For each source, we have a set of candidate values V, and for each of the candidate values $v \in V$ a set of supporting evidence sources S_v. In the case of a knowledge base, the triples where the predicate matches the target heading h or related headings h' are the evidence sources. In the case of a table corpus, S_v contains all candidate tables T' where v is a cell value corresponding the target entity e and to heading label h'. Our task is to score each candidate value v based on the available evidence.

There are two main challenges we need to deal with. One is how to handle `Empty` values, i.e., quantify our confidence in that the given cell should be left empty. Another is how to combine evidence across multiple sources, specifically, a knowledge base and a table corpus. We start by considering each source individually in Sects. 5.1 and 5.2, and then combine the two in a feature-based learning approach in Sect. 5.3. Novel contributions in this part include the source-specific value finding methods as well as the three groups of features, each designed with a specific intuition in mind: (i) quantifying the support each source has for a given value, (ii) dealing with `Empty` values, and (iii) effectively prioritizing values from semantically more related tables in a table corpus.

5.1 Knowledge Base

We deal with empty values by adding a designated special value `Empty` to the set of candidates. The scoring of values is then based on the following formula:

$$score(v; e, h, T) = \begin{cases} \arg\max_p score(p, h), & v \neq \texttt{Empty} \\ \gamma, & v = \texttt{Empty}\,, \end{cases} \qquad (3)$$

where γ is a free parameter that we learn empirically. For non-empty values, $score(p, h)$ can be estimated in two alternative ways:

- We take the edit distance between the column heading and the (label of the) predicate[1] (referred to as soft matching in [25]).

$$score_{ED}(p, h) = 1 - \frac{dist(p, h)}{\max(|p|, |h|)}\,, \qquad (4)$$

  where $dist()$ represents the minimum number of single-character edit operations (i.e., insertion, deletion, or substitution) needed to transform one string into another.
- We use the conditional probability $P(p|h)$ (cf. Eq. (2)).

5.2 Table Corpus

A given candidate value may have multiple supporting tables in the corpus. We formulate two evidence combination strategies. One method (§5.2.1) is to consider a single table that best matches semantically the input table, similarly to [1]. Another method (§5.2.2) is to consider multiple tables but weigh them according on their semantic similarity to the input table, an idea in line with [20, 22]. Both methods are based on the notion of *table matching*, which we described in §4.1.2. One important addition, compared to prior approaches, is that we also consider which heading h' of the candidate table T' matches best the target heading h (§5.2.3). As we will show in our experiments, this has a positive effect.

5.2.1 Top-ranked Table. This method takes the best matching candidate table T', i.e., the one that is most similar to the input table T, and combines that table's matching score with the best matching heading label within that table. Formally:

$$score(v, e, h, T) = \arg\max_{T'} score(T', T) \times \big(\max_{h' \in T'} sim(h', h)\big)\,. \qquad (5)$$

5.2.2 All Tables. Alternatively, one might consider all matching tables, as opposed to a single best one, and aggregate information from these. Formally:

$$score(v, e, h, T) = \sum_{T'} \Big(score(T', T) \times \max_{h' \in T'} sim(h', h)\Big)\,, \qquad (6)$$

where $sim(h', h)$ is the similarity between two heading column labels, which we detail below.

5.2.3 Heading Label Similarity. We consider four methods for computing the similarity $sim(h', h)$ between two heading column labels:

- *Uniform*: we set similarity to a fixed value (e.g., 1). This way the similarity of headings is not considered at all in Eqs. (5) and (6). (The uniform estimator will merely serve as a baseline, to evaluate the benefits of incorporating heading similarity.)
- *Edit distance*: We use the edit distance between h and h' (as in Eq. (4), but replacing p with h').
- *Mapping probability*: we use the conditional probability $P(h'|h)$ as defined in Eq. (1).
- *Label2Vec*: We employ the skip-gram model of Word2vec [13] to train heading label embeddings on the table corpus. Then, $sim(h', h)$ is taken to be the cosine similarity between the embedding vectors of h' and h, respectively.

5.3 Combination of Evidence

We combine evidence from multiple sources using a feature-based approach. Table 3 summarizes our features. Additionally, we use the same table quality/importance features as for table matching (cf. top block in Table 1).

Feature group I captures how much support there is for the given value in each source. Two binary features (IS_TC and IS_KB) are meant to indicate whether the value can be found in a given source (table corpus and knowledge base). The next four features are used for capturing heading level similarity, based on edit distance (EDITDIST_PH and EDITDIST_HH) and mapping probability (MAPPINGPROB_PH and MAPPINGPROB_HH).

Feature group II aims at empty value prediction. The intuition is that if the entity-heading combination appears a lot in the table corpus or in the knowledge base, then we have a better chance of finding a value. Some features (NUM_E, NUM_H, and NUM_EH) are general statistics on the number of entity, heading or entity-heading occurrences in the tables corpus. EMPTY_RATE measures the fraction of cells that are empty in a given column. MATCH_EH_NUM and MATCH_HH_NUM are the number of predicate-heading and heading-heading matches. The last 6 features are the aggregated

[1]We are not using the predicate itself, but the corresponding label from the KB that is meant for human consumption. E.g., for `<dbp:timeZone>` the label is "time zone."

Table 3: Features for value ranking. e and h denote the entity and heading column, respectively in the input table T, while h' is the best matching column (based on mapping probability) in the candidate table T'.

Group / Feature	Description	Source	Value
Feature group I			
IS_TC	Whether the value comes from the table corpus ($v \in V_{TC}$)	TC	$\{0, 1\}$
IS_KB	Whether the value comes from the knowledge base ($v \in V_{KB}$)	KB	$\{0, 1\}$
EDITDIST_PH	Predicate-to-heading edit distance ($score_{ED}(p, h)$)	KB, TC	$[0, 1]$
MAPPINGPROB_PH	Predicate-to-heading mapping probability ($P(p\|h)$)	KB, TC	$[0, 1]$
EDITDIST_HH	Heading-to-heading edit distance ($score_{ED}(h', h)$)	TC	$[0, 1]$
MAPPINGPROB_HH	Heading-to-heading mapping probability ($P(h'\|h)$)	TC	$[0, 1]$
Feature group II			
NUM_E	Number of times entity e appears in the table corpus	TC	$[0, \infty)$
NUM_H	Number of times heading h appears in the table corpus	TC	$[0, \infty)$
NUM_EH	Number of times entity e and heading h co-occur in the table corpus	TC	$[0, \infty)$
EMPTY_RATE	Fraction of empty cells in column h in the table corpus	TC	$[0, 1]$
MATCH_PH_NUM	Number of predicate-to-heading matches ($\|\{p : P(p\|h) > 0\}\|$)	KB, TC	$[0, \infty)$
MATCH_HH_NUM	Number of heading-to-heading matches ($\|\{h' : P(h'\|h) > 0\}\|$)	TC	$[0, \infty)$
MATCH_PH_{MAX,AVG,SUM}	Aggregated number of predicate-to-heading matches ($aggr_p\big[count(p, h)\big]$)	KB, TC	$[0, \infty)$
MATCH_HH_{MAX,AVG,SUM}	Aggregated number of heading-to-heading matches ($aggr_{h'}\big[count(h', h)\big]$)	TC	$[0, \infty)$
Feature group III			
TMATCH_NUM	Number of matching tables, using TMatch scorer ($\|\{T' : score_{HCF}(T', T) > 0\}\|$)	TC	$[0, \infty)$
TMATCH_{MAX,AVG,SUM}	Aggregated table matching scores, using TMatch scorer ($aggr_{T'}\big[score_{HCF}(T', T)\big]$)	TC	$[0, \infty)$
SCORE_IG_ED_{MAX,AVG,SUM}	Aggregated value score using InfoGather table matching with edit distance	TC	$[0, \infty)$
SCORE_IG_MP_{MAX,AVG,SUM}	Aggregated value score using InfoGather table matching with mapping probability	TC	$[0, \infty)$
SCORE_IG_L2V_{MAX,AVG,SUM}	Aggregated value score using InfoGather table matching with Label2vec	TC	$[0, \infty)$
SCORE_TMATCH_ED_{MAX,AVG,SUM}	Aggregated value score using TMatch table matching with edit distance	TC	$[0, \infty)$
SCORE_TMATCH_MP_{MAX,AVG,SUM}	Aggregated value score using TMatch table matching with mapping probability	TC	$[0, \infty)$
SCORE_TMATCH_L2V_{MAX,AVG,SUM}	Aggregated value score using TMatch table matching with Label2vec	TC	$[0, \infty)$

counts of predicate-heading and heading-heading matches in the knowledge base and in the table corpus.[2]

Feature group III aims for capturing the semantic relatedness between the input table and candidate table where the value is taken from. The matches between the input table and candidate tables are captured in the number of matching tables (TMATCH_NUM) as well as aggregates over the table matching scores (TMATCH_*). Additionally, we consider the value scoring mechanism devised specifically for TC (cf. Sect. 5.2), which involves a table matching method (InfoGather (IG) or TMatch), heading similarity (edit distance (ED), mapping probability (MP), or Label2vec (L2V)), and an aggregator (MAX, AVG, or SUM). All possible combinations yield a total of 18 features. For example, SCORE_IG_ED_SUM corresponds to Eq. (6) using InfoGather for table matching and edit distance heading similarity, and SCORE_TMATCH_L2V_MAX corresponds to Eq. (5) using TMatch table matching, and Label2Vec heading similarity.

6 EXPERIMENTAL SETUP

Auto-completion for data cells is a novel problem, and as such, no public test collection exists. In this section, we introduce the data sources used in our experiments and describe the construction of our test collection, which is another main contribution of this study. It is based on 1000 table cells and contains labels for 35k cell-value pairs, obtained via crowdsourcing. We also present the techniques we employed for column data type detection and value normalization, which are essential to ensure data quality.

[2]For predicate-to-heading and heading-to-heading matching, empty values are not considered, i.e., two empty cell values are not considered as being the same.

6.1 Data Sources

We use two main data sources:

- **Table Corpus (TC)** The WikiTables corpus [4] is extracted from Wikipedia and contains 1.6M high-quality tables. Following [24], we select the core column by taking the one among the left-most 2 columns with the highest entity rate. Based on a sample of 100 tables, this method has over 98% accuracy. There are 755k relational tables in the corpus that have a core column where 80% of the cell values are entities. Since tables are from Wikipedia, the mentioned entities have been explicitly marked up.
- **Knowledge Base (KB)** DBpedia is a general-purpose knowledge base that is a central hub in the Linked Open Data cloud. Specifically, we use the 2015-10 version and restrict ourselves to entities that have at least a short description (abstract), amounting to a total of 4.6M entities.

6.2 Column Data Type Detection

Our objective is to classify a given table column according to some taxonomy of data types. It is assumed that all cells within a column share the same data type. To determine the data type of a given column, we classify each (non-empty) cell within that column and then take a majority vote. In the rare case of a tie, the column will be assigned multiple types. In the following, we introduce the value data type taxonomy used and our method for classifying the value data types of individual table cells.

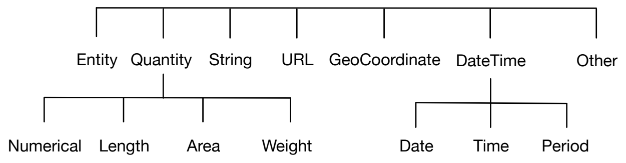


Figure 4: Value data type taxonomy used in this paper.

6.2.1 *Value Data Type Taxonomy.* Different value data type taxonomies have been proposed in the literature, see, e.g., [16, 21]. We build on and extend the taxonomy by Yin et al. [21], who consider seven value data types in the context of fact finding from web tables: string, date/time, numerical, duration, length, area, and weight. Knowledge bases also have their own data type taxonomies, e.g., DBpedia has 25 data types.[3] Informed by these, we introduce a two-layer value data type taxonomy, which is shown in Figure 4. We manually map the data types of the knowledge base (here: DBpedia) to our value data type taxonomy.

6.2.2 *Cell Value Type Classification.* Following standard practice [16], we design a rule-based method for classifying cell values into our value data type taxonomy. For example, DateTime values are identified based on the cell values matching given patterns and on certain terms appearing in the column heading label (such as "year," "birth," "date," "founded," "created," or "built"). The complete set of rules is released in the online appendix.

6.2.3 *Evaluation.* We obtain the distribution of column data types based on a sample of 100k relational tables from the table corpus. The results, ordered by frequency, are as follows: (1) Quantity: 269,260 cells (43.2%), (2) Entity: 200,637 cells (32.2%), (3) String: 76,259 cells (12.2%), (4) Other: 51,999 cells (8.3%), (5) DateTime: 24,413 cells 3.9%, (6) GeoCoordinate: 161 cells (0.0%). There were not any cells of type URL, as in our sample all links refer to entities in Wikipedia. Given that number of columns with type GeoCoordinate is negligible, we exclude this type in our experiments. We further note that columns with type Other contain mostly empty values.

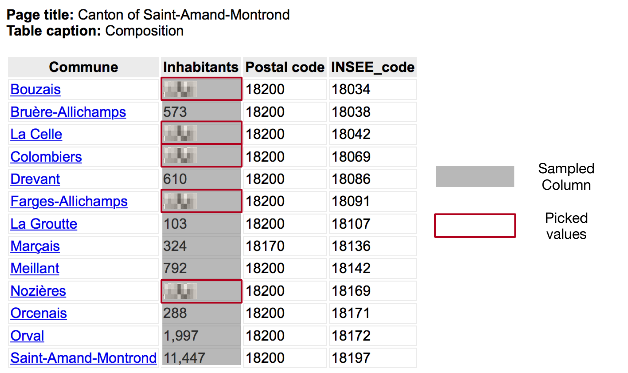
To verify whether the performance of our column type detection method is sufficient, we manually evaluate it on a sample of 100 tables. Specifically, tables are selected such that each has at least 6 rows and 4 columns, and has a core column where over 80% of the cell values are entities. Our sample contains a total of 473 table columns. The accuracy of column type detection is found to be 94.92%.

6.3 Value Normalization

The previous step informs us about the data type of the value that we are looking for. Values, however, may be expressed in a variety of ways in different sources. For example, dates are written differently by individuals in different parts of the world. We normalize cell values according to the data types of the corresponding column.

To ensure high data quality, we employ a rule-based approach. On close inspection of the data, we develop over 100 rules for normalizing cell values based on their data types. We illustrate these transformations with some examples. All dates are converted to "YYYY-MM-DD" format and all times are transformed to "HH:MM:SS" format. Date periods with only years are normalized to "year–year" and those with dates are separated into two dates. E.g., "1998–99" is normalized to "[1998,1999]," while "5 October 1987 to 30 December 1987" is converted to "[1987-10-05, 1987-12-30]." For quantities, the numeric values and the units are kept separately, e.g, "100 m" is stored as (100, "m") and "-54 kilograms" is stored as (-54, "kilograms"). No unit conversion is performed. In the case of composite values, we only keep the first value, e.g., "71 kg/m^2 (14.5 lb/ft^2)" is stored as (71, "kg/m^2").

[3] http://mappings.dbpedia.org/index.php/DBpedia_Datatypes



Page title: Canton of Saint-Amand-Montrond
Table caption: Composition

Commune	Inhabitants	Postal code	INSEE_code
Bouzais	[illegible]	18200	18034
Bruère-Allichamps	573	18200	18038
La Celle	[illegible]	18200	18042
Colombiers	[illegible]	18200	18069
Drevant	610	18200	18086
Farges-Allichamps	[illegible]	18200	18091
La Groutte	103	18200	18107
Marçais	324	18170	18136
Meillant	792	18200	18142
Nozières	[illegible]	18200	18169
Orcenais	288	18200	18171
Orval	1,997	18200	18172
Saint-Amand-Montrond	11,447	18200	18197

Figure 5: Illustration of sampling values. First, the column of "Inhabitants" is the sampled as a numerical column. Then, five values in this column are picked randomly as test cases.

Table 4: Test collection statistics (based on 1000 cells).

	KB	TC	KB+TC
Avg. #values	1.31	2.51	2.65
Empty rate	0.48	0.31	0.18

6.4 Test Collection

We create a test collection for value finding based on a sample of existing tables from the table corpus. (These test tables are excluded from our index and when computing statistics.) Specifically, we perform stratified sampling according to the four main column data types: Entity, Quantity, String, and DateTime. For each data type, we first randomly select 50 columns, each from a different table, where there is at least 80% agreement on the column data type according to the majority vote method (cf. Sect. 6.2.2). We further require that the table has at least 5 rows and 3 columns, and the respective heading label has a certain minimum length (4 characters). From each sampled table column, 5 specific cells are picked randomly. This way, our test collection consists of $4 \times 50 \times 5 = 1000$ cells for which we are trying to find values. These input tables are then excluded from the collection. See Fig. 5 for an illustration.

Relevance assessments were collected via crowdsourcing using the Figure Eight platform.[4] For each cell, human assessors were presented with the page title (embedding the table), table caption, the core column entity, the heading column label, and a source document. The source document is either the DBpedia page of the core column entity or an existing table from the table corpus. Users were then asked to check if the missing cell value can be found within the source document, and, if yes, to provide the corresponding value (otherwise enter a designated special `Empty` value).

[4] https://www.figure-eight.com/

Table 5: Value finding performance. Significance for line i ($i > 1$) is tested against the best method in lines $1..i-1$.

Source	Method	Sources used		`Empty` excluded		`Empty` included	
		KB	TC	NDCG@5	NDCG@10	NDCG@5	NDCG@10
Single-source	KBLookupED	✓		0.2635	0.2652	0.2780	0.2806
	InfoGather, top, UNI		✓	$0.4563^‡$	$0.4710^‡$	$0.4158^‡$	$0.4302^‡$
	InfoGather, top, L2V		✓	0.4868	0.4978	0.4413	0.4537
	TMatch, top, UNI		✓	$0.4744^‡$	$0.4873^‡$	$0.4297^‡$	$0.4417^‡$
	TMatch, top, L2V		✓	0.5046	0.5139	0.4531	0.4624
Multi-source	OTG [25]	✓	✓	0.5856	0.6062	0.5185	0.5367
	CellAutoComplete (feat. I)	✓	✓	$0.6641^‡$	$0.6826^‡$	$0.5766^‡$	$0.5954^‡$
	CellAutoComplete (feat. I+II)	✓	✓	$0.6844^‡$	$0.7034^‡$	$0.5905^‡$	$0.6100^‡$
	CellAutoComplete (feat. I+II+III)	✓	✓	**$0.7570^‡$**	**$0.7641^‡$**	**$0.6716^‡$**	**$0.6785^‡$**

Each instance was annotated by 7 assessors. The inter-annotator agreement in terms of Fleiss' kappa statistic was about 0.7 when using the knowledge base and 0.8 when using the table corpus as source. The former is considered as substantial, the latter is considered as almost perfect agreement [11]. The total expense of the crowdsourcing experiments was $770.

We then combine the correct values from these two sources as our ground truth. (We only use the KB and TC specific subsets in our analysis of specific sources in Sect. 7.2.) Table 4 shows statistics of our test collection. We find that, when using both sources, cells on average have over two possible correct values. It is further worth noting that the rate of empty cells is much lower when combining the two sources, attesting to their complementary nature.

6.5 Table Matching

To train our table matching models (InfoGather and TMatch in Sect. 4.1.2), we construct a training dataset. We group tables by topics and sample 50 tables with diverse topics (such as military, paleontology, sports, geography, etc.) from the corpus as input tables. Each table should have at least five rows and three columns. For each table, we utilize the query-based search methods in [1] to obtain a set of candidate tables. We ask 3 annotators to judge if the candidate table is highly relevant, relevant, or not relevant.

6.6 Evaluation Measures

We evaluate performance in terms of Normalized Discounted Cumulative Gain (NDCG) at cut-off points 5 and 10. To test significance, we use a two-tailed paired t-test and write †/‡ to denote significance at the 0.05 and 0.01 levels, respectively.

7 EXPERIMENTAL EVALUATION

This section presents evaluation results for the value finding task (Sect. 7.1) followed by further analysis of value sources (Sect. 7.2), features (Sect. 7.3), and specific examples (Sect. 7.4).

7.1 Evaluating Auto-Completion

We begin with the evaluation of the end-to-end cell value auto-completion task. Table 5 reports the results. At the top block of Table 5, we display the methods that use an individual source, either knowledge base (KB, line 1) or table corpus (TC, lines 2–5). These methods meant to serve as single-source baselines; they are further detailed in Sect. 7.2. The bottom block of Table 5 shows methods that utilize both sources. There is only one existing work in the literature that we found directly applicable: the On-the-Fly Table Generation (OTG) approach by Zhang and Balog [25]. This method combines a knowledge base and a table corpus in a simple way, by always giving preference to the former source over the latter.

Looking at the results in Table 5, it is clear that the table corpus is a more effective source for value finding than the knowledge base. At the same time, they are complementary and combining the two yields substantial improvements. This is already witnessed for OTG [25], but to a much larger extent with our CellAutoComplete methods. Our best methods, using the complete feature set (cf. Table 3) outperforms OTG substantially, i.e., by over 26% on all evaluation metric and experimental conditions (lines 6 vs. 9). These improvements can be attributed to two main factors. First, instead of naively giving preference to the knowledge base over the table corpus, as in OTG, CellAutoComplete (feat. I) decides for each cell individually which source should be preferred, by considering the predicate-to-heading and heading-to-heading matching probabilities, among other signals. This makes a large difference, as can be observed in the scores (lines 6 vs. 7). Second, taking into account the semantic similarity of tables, when using a table corpus as source, makes a large difference. This is what feature group III contributes. We find that it brings in an over 10% relative improvement, see CellAutoComplete (feat. I+II) vs. (feat. I+II+III), i.e., the bottom two lines in Table 3. As for the second group of features, which aims at improving empty value prediction, we find that is has a small, but positive and significant impact (feat. I vs. I+II).

7.2 Analysis of Sources

Next, we analyze cell auto-completion performance using only a single source: a knowledge base (Table 6) and a table corpus (Table 7). As before, we distinguish between two settings, with `Empty` values excluded and included. We note that the ground truth is restricted to the specific source, therefore, it is different in the two cases (and also different from Table 5, which uses the union of the two).

7.2.1 Using a Knowledge Base. We compare two different KB-based value lookup methods, edit distance (ED) and matching probability (MP), in Table 6. The two methods yield virtually identical performance when `Empty` values are excluded. When `Empty` values are considered, ED performs significantly better than MP. Recall that

Table 6: Value finding performance using a knowledge base. Significance of MP is tested against ED.

Method	Empty excluded		Empty included	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
KBLookup ED	**0.5255**	0.5308	**0.5015**	**0.5048**
KBLookup MP	0.5222	**0.5316**	0.4489‡	0.4549‡

our approach involves a γ threshold for `Empty` detection (cf. Eq. (3)). Here, we estimate this threshold using 5-fold cross-validation, and the average γ value is 0.8 for ED and 0.6 for MP. The reason that edit distance performs better is that it is more robust with respect to the value of γ. In other words, a single γ value performs well across different predicate-column heading pairs.

7.2.2 Using a Table Corpus. We consider (i) two table matching methods, InfoGather and TMatch;[5] (ii) two evidence combination strategies, top-ranked table (top) and all tables (all); and (iii) four heading label similarity methods, uniform (UNI), edit distance (ED), mapping probability (MP), and Label2Vec (L2V). Table 7 presents all possible combinations of these.

Our observations are as follows. Regarding the two table matching methods (lines 1–8 vs. 9–16), we find that TMatch can outperform InfoGather by up to 18%, with all other components being identical. Many of the differences (esp. when using all tables) are statistically significant. This shows that value finding benefits from better table matching, which is as expected. When comparing the two evidence combination strategies (lines 1–4 vs. 5–8 and 9–12 vs. 13–16), we find the *top* method to be the better overall performer. There are a few exceptions, however, when *all* delivers marginally better results, e.g., TMatch with ED, MP, or L2V, with `Empty` included. Finally, the ranking of heading label similarity methods is ED, L2V > UNI > MP. That is, ED and L2V perform best, with minor differences between the two depending on the particular configuration. Interestingly, MP does not work well, in fact, it performs even worse than not incorporating heading similarity at all (UNI).

7.3 Feature Importance Analysis

In order to gain an understanding of which features contribute most to the effectiveness of our value ranking approach, we measure their importance in terms of Gini score. The results are shown in Fig. 6, ordered left to right from most to least important. Generally, features from group I and III are the most represented at the top ranks, while feature group II and table features dominate the bottom half of the ranking.

7.4 Cell-level Analysis

So far, we have reported on aggregate statistics. In our final experimental section, we perform an analysis on the level of individual cells. Recall that when creating the test collection, we have concealed the original cell values from the input tables, pretending that these were missing. In this part, we compare these original cell values (referred to as *original*) with the values that were retrieved automatically by our approach (referred to as *found*).

[5]Additionally, we have also considered DTS [14] and the method in [6] for table matching. However, both were inferior to InfoGather in terms of effectiveness. Therefore, in the interest of space we only report only on InfoGather.

Table 7: Value finding performance using a table corpus. Highest score are boldfaced. Significance of TMatch (lines 9-16) is tested against InfoGather (lines 1-8).

Method			Empty excluded		Empty included	
			NDCG@5	NDCG@10	NDCG@5	NDCG@10
InfoGather	top	UNI	0.6178	0.6425	0.5142	0.5370
		ED	**0.6670**	**0.6854**	**0.5497**	**0.5675**
		MP	0.4968	0.5428	0.4474	0.4848
		L2V	0.6600	0.6792	0.5442	0.5634
InfoGather	all	UNI	0.5992	0.6255	0.5052	0.5289
		ED	0.6445	0.6685	**0.5561**	**0.5753**
		MP	0.4677	0.5252	0.4348	0.4802
		L2V	**0.6489**	**0.6714**	0.5365	0.5576
TMatch	top	UNI	0.6463‡	0.6670‡	0.5342†	0.5524‡
		ED	**0.6930**‡	**0.7077**‡	0.5626	0.5772
		MP	0.5256‡	0.5790‡	0.4664	0.5096
		L2V	0.6863‡	0.7028‡	**0.5630**‡	**0.5791**‡
TMatch	all	UNI	0.6208‡	0.6459‡	0.5335‡	0.5534‡
		ED	0.6402	0.6692	0.5534	0.5753
		MP	0.5234	0.5427‡	0.4788	0.4921
		L2V	**0.6739**‡	**0.6921**‡	**0.5678**‡	**0.5851**‡

Table 8: Cell-level analysis, comparing the original (concealed) values in test tables against those found by our CellAutoComplete method.

Found / Original	**0**	**1**	**2+**
0	20	3	5
1	154	205	613

Table 8 reports the overall statistics. The first and second lines of this table represent the cases where the cell was originally empty and had a value in the input table, respectively. The columns of the table correspond to how many different (valid) values were found by our approach. Below, we take a closer look at each of these cases, from top to bottom and from left to right.

- There are 20 cells, where originally the cell was empty and we also did not find a value (i.e., no difference).
- In 3 cases, we found the value for a cell that was originally empty. On such example is the "departure" time for "Hampton Roads" in a table about "Itinerary."
- In 5 cases, we identified two valid values for a cell that was originally empty. For instance, for the "type" column of "Polvorones", in a table about "Breads and pastries," both "shortbread" and "bread" are correct values.
- There are 154 cells where the cell is originally not empty, but we could not find its value. This is the category where our method failed. It turns out that in most of these cases, the given values exist only in the original tables (which were excluded from the corpus).
- In 205 cases, both *original* and *found* have the same single value (i.e., no difference).
- For 613 cells that are originally non-empty, we found multiple valid values. In many cases, *found* includes further values in

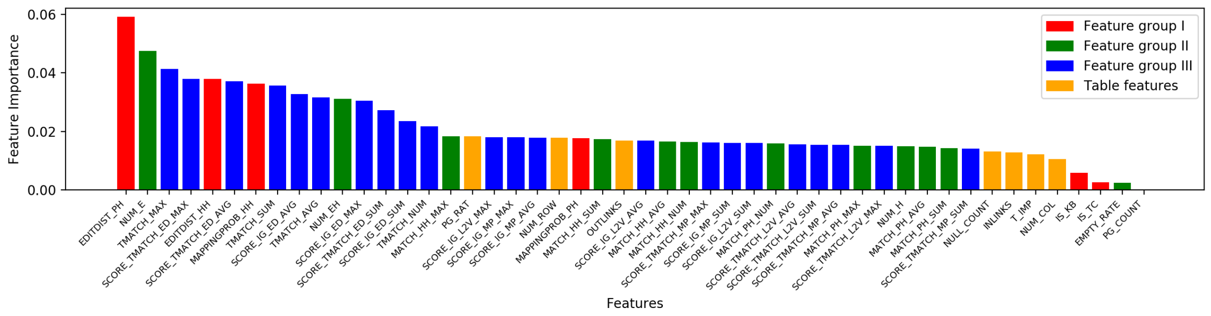


Figure 6: Feature importance measured in terms of Gini score.

addition to the original value. E.g., the original value is "Republican Party (United States)," while the found values also include "R" and "Republican." Another example is an athlete's "country," which is originally "Brazil at the Olympics", while the found values also include "Brazil." In some cases the granularity of the values differ, e.g., the "location" of "Pike" is "Levee Township, Pike County, Illinois" in the original table, while values in *found* also include "Hull, Illinois", "Detroit, Illinois", "Pittsfield, Illinois", and "Pearl, Illinoi." In other cases, there is no overlap between the values returned by *original* and *found*. There are several cases where the difference is in the value formats or in the granularity. E.g., the original table contains "1982" as the "death" date of "Hugh John Flemming," while the value we returned from the knowledge base is "1982-10-16." Another reason for the differences has to do with temporal mismatch, i.e., one of the sources is out-of-date.Finally, there are also some genuine cases of conflicting values. E.g., the "open date" of "Kannon Station" is "1923-07-05" in the original table, while in DBpedia the "opening year" is "1913-01-01." Similarly, the "Platform" of "Okular" is "MS" according to one Wikipedia table, while it is "Unix-like" in DBpedia.

Overall, our method finds the same as the original value in 22.5% of the cases, misses the original value in 15.4% of the cases, and finds either additional correct values or conflicting values in 62.1% of the cases. This latter category highlights the usefulness of cell auto-completion. It also suggests further potential for other applications, such as fact-checking.

8 CONCLUSIONS

We have addressed the task of auto-completing cell values, given an input relational table. Using a knowledge base and a table corpus as sources, we have demonstrated the effectiveness of our approach on a purpose-built test collection. While we have developed our approach with a specific application in mind, these techniques can also be utilized for other tasks, including information extraction, populating KBs from tables, and truth/fact finding.

We see several avenues for future work. We would like to move beyond relational tables and beyond the clean and well-organized tables that can be found in Wikipedia, by considering arbitrary tables from the Web. In addition to structured sources, we are interested in incorporating evidence from unstructured text, e.g., web pages. Finally, we wish to explicitly address the temporal aspects of certain entity attributes.

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] Marie Anan and Gal Avigdor. 2007. On the Stable Marriage of Maximum Weight Royal Couples. In *Proc. of IIweb '07*. 1–6.

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

[4] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proc. of ISWC '15*. 425–441.

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

[6] 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*.

[7] Li Deng, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *Proc. of SIGIR'19*.

[8] Xin Dong and Alon Y. Halevy. 2005. Malleable Schemas: A Preliminary Report. In *Proc. of the WebDB '05*.

[9] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings. In *Proc. of ISWC '17*.

[10] Patrick Ernst, Amy Siu, and Gerhard Weikum. 2018. HighLife: Higher-arity Fact Harvesting. In *Proc. of WWW '18*. 1013–1022.

[11] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33 (1977).

[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] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proc. of NIPS '13*. 3111–3119.

[14] Thanh Tam Nguyen, Quoc Viet Hung Nguyen, Weidlich Matthias, and Aberer Karl. 2015. Result Selection and Summarization for Web Table Search. In *ISDE '15*. 231–242.

[15] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval. *Inf. Retr.* 13, 4 (Aug 2010), 346–374.

[16] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *Proc. of WIMS '15*. 10:1–10:6.

[17] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table Cell Search for Question Answering. In *Proc. of WWW '16*. 771–782.

[18] 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. of VLDB Endow.* 4 (2011), 528–538.

[19] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[20] 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.

[21] Xiaoxin Yin, Wenzhao Tan, and Chao Liu. 2011. FACTO: A Fact Lookup Engine Based on Web Tables. In *Proc. of WWW '11*. 507–516.

[22] 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.

[23] Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proc. of SIGIR '17*. 255–264.

[24] Shuo Zhang and Krisztian Balog. 2018. Ad Hoc Table Retrieval using Semantic Similarity. In *Proc. of WWW '18*. 1553–1562.

[25] Shuo Zhang and Krisztian Balog. 2018. On-the-fly Table Generation. In *Proc. of the SIGIR '18*. 595–604.