Towards Efficient and Effective Semantic Table Interpretation

Ziqi Zhang

Department of Computer Science, University of Sheffield, UK
z.zhang@dcs.shef.ac.uk

Abstract. This paper describes TableMiner, the first semantic Table Interpretation method that adopts an incremental, mutually recursive and bootstrapping learning approach seeded by automatically selected 'partial' data from a table. TableMiner labels columns containing named entity mentions with semantic concepts that best describe data in columns, and disambiguates entity content cells in these columns. TableMiner is able to use various types of contextual information outside tables for Table Interpretation, including semantic markups (e.g., RDFa/microdata annotations) that to the best of our knowledge, have never been used in Natural Language Processing tasks. Evaluation on two datasets shows that compared to two baselines, TableMiner consistently obtains the best performance. In the classification task, it achieves significant improvements of between 0.08 and 0.38 F1 depending on different baseline methods; in the disambiguation task, it outperforms both baselines by between 0.19 and 0.37 in Precision on one dataset, and between 0.02 and 0.03 F1 on the other dataset. Observation also shows that the bootstrapping learning approach adopted by TableMiner can potentially deliver computational savings of between 24 and 60% against classic methods that 'exhaustively' processes the entire table content to build features for interpretation.

1 Introduction

Recovering semantics from tables on the Web is becoming a crucial task towards realizing the vision of Semantic Web. On the one hand, the amount of high-quality tables containing useful relational data is growing rapidly to hundreds of millions [5, 4]; on the other hand, classic search engines built for unstructured free-text perform poorly on such data as they ignore the underlying semantics in table structures at indexing time [12, 16]. Semantic **Table Interpretation** [12, 21–23, 16] aims to address this issue by solving three tasks: given a well-formed relational table[1] and a knowledge base defining a set of reference concepts and entities interlinked by relations, 1) recognize the semantic concept (or a property of a concept) that best describes the data in a column (i.e., classify columns); 2) identify the semantic relations between columns (i.e., relation enumeration); and 3) disambiguate content cells by linking them to existing (if any) entities in the knowledge base (i.e., entity disambiguation). Essentially, the knowledge

[1] Same as others, this work assumes availability of well-formed relational tables while methods of detecting them can be found in, e.g., [5]. A typical relational table is composed of regular rows and columns resembling those in traditional databases.

base is a linked data set where resources are connected as triples. The outcome of semantic Table Interpretation is semantically annotated tabular data, which does not only enable effective indexing and search of the data, but ultimately can be transformed to new triples (e.g., new instances of concepts and relations) to populate the Linked Open Data (LOD) cloud.

The tasks resemble the classic Natural Language Processing tasks that have been extensively researched for decades, i.e., Named Entity Classification [18], Relation Extraction [19] and Named Entity Disambiguation [7]. However, classic approaches often fails at tabular data since they are trained for well-formed, unstructured sentences which are rare in table structures. Semantic Table Interpretation methods [12, 21–23, 16] typically depend on background knowledge bases to build features for learning. The typical workflow involves 1) retrieving candidates matching table components (e.g., a column header) from the knowledge base, 2) constructing features of candidates and model semantic interdependence between candidates and table components, and among various table components, and 3) applying inference to choose the best candidates.

This paper introduces TableMiner, designed to classify columns and disambiguate the contained cells in an unsupervised way that is both efficient and effective, addressing two limitations in existing works. First, existing methods have predominantly adopted an *exhaustive* strategy to build the candidate space for inference, e.g., column classification depends on candidate entities from *all* cells in the column [12, 16]. However, we argue this is unnecessary. Consider the table shown in Figure 1 as a snapshot of a rather large table containing over 50 rows of similar data. One does not need to read the entire table in order to label the three columns. Being able to make such inference using *partial* (as opposed to the entire table) data can improve the efficiency of Table Interpretation algorithms as the first two phases in the Table Interpretation workflow can cost up to 99% of computation time [12]. Second, inference algorithms of state-of-the-art are almost exclusively based on two types of features: those derived from background knowledge bases (in generic form, triples from certain linked data sets) and those derived from table components such as header text, and row content. This work notes that the document context that tables occur in (i.e., *around* and *outside* tables e.g., captions, page titles) offers equally useful clues for interpretation. In particular, another source of linked data - the pre-defined semantic markups within Webpages such as RDFa/microdata[2] annotations - provide important information about the Webpages and tables they contain. However such data have never been used in Table Interpretation tasks, even not in any NLP tasks in general.

TableMiner adopts a two-phase incremental, bootstrapping approach to interpret columns. A *forward-learning* phase uses an *incremental inference with stopping* algorithm (*I-inf*) that builds initial interpretation on an iterative row-by-row basis until TableMiner is 'sufficiently confident' (automatically determined by convergence) about the column classification result. Next, a *backward-update* phase begins by using initial results from the first phase (seeds) to constrain and guide interpretation of the remaining data. This can change the classification results on a column due to the newly disambiguated entity content cells. Therefore it is followed by a process to update classification and disambiguation results in the column in a mutually recursive pattern until

[2] E.g., with the schema.org vocabulary

they become stabilized. In both tasks, TableMiner uses various types of table context (including pre-defined semantic markups within Webpages where available) to assist interpretation.

Evaluation on two datasets shows that TableMiner significantly outperforms two baselines in both classification (between 0.08 and 0.38 in F1) and disambiguation (between 0.19 and 0.37 Precision on one dataset based on manual inspection, and 0.02 to 0.03 F1 on another) tasks, and offers substantial potential to improve computational efficiency.

The remainder of this paper is organized as follows: Section 2 discusses related work, Section 3 introduces the methodology, Section 4 describes evaluation and discusses results, and Section 5 concludes this paper.

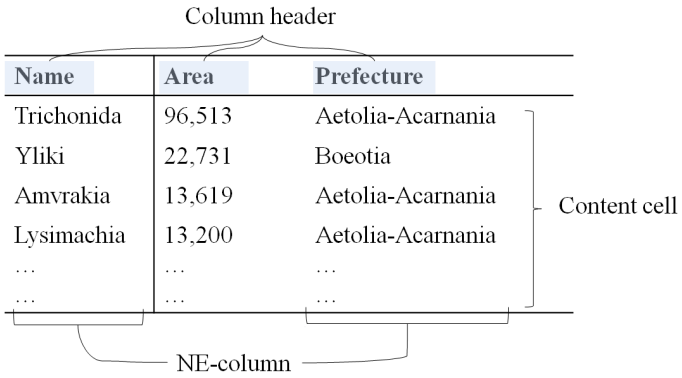

Fig. 1. Lakes in Central Greece (adapted from Wikipedia)

2 Related Work

This work belongs to the general domain of table information extraction covering a wide range of topics such as table structure understanding [25] that aims to uncover structural relations underlying table layout in complex tables; relational table identification that aims to separate tables containing relational data from noisy ones used for, e.g., page formatting, and then subsequently identifying table schema [5, 4, 1]; table schema matching and data integration that aims to merge tables describing similar data [2, 3, 13]; and semantic Table Interpretation, which is the focus of this work. It also belongs to the domain of (semi-)structured Information Extraction, where an extensive amount of literature is marginally related.

2.1 Semantic Table Interpretation

Venetis et al. [22] annotate columns in a table with semantic concepts and identify relations between the subject column (typically containing entities that the table is about)

and other columns using a database mined with regular lexico-syntactic patterns such as the Hearst patterns [9]. The database records co-occurrence statistics for each pair of values extracted by such patterns. A maximum likelihood inference model is used to predict the best concepts and relations from candidates using these statistics.

Similarly, Wang et al. [23] first identify a subject column in the table, then based on subject entity mentions in the column and their corresponding values in other columns, associate a concept from the Probase knowledge base [24] that best describes the table schema (hence properties of the concept are used to label the columns). Essentially this classifies table columns and identifies relations between the subject column and other columns. Probase is a probabilistic database built in the similar way as that in Venetis et al. [22] and contains an inverted index that supports searching and ranking candidate concepts given a list of terms describing possible concept properties, or names describing possible instances. Interpretation heavily depends on these features and the probability statistics gathered in the database.

Limaye et al. [12] use factor graph to model a table and the interdependencies between its components. Table components are modeled as variables represented as nodes on the graph; then the interdependencies among variables and between a variable and its candidates are modeled by factors. The task of inference amounts to searching for an assignment of values to the variables that maximizes the joint probability. A unique feature of this method is it addresses all three tasks simultaneously. Although the key motivation is using joint inference about each of the individual components to boost the overall quality of the labels, later study showed that this does not necessarily guarantee advantages over models that address each task separately and independently [22]. Furthermore, Mulwad et al. [16] argue that computing the joint probability distribution in the model is very expensive. Thus built on their earlier work by [21, 17, 15], they introduce a semantic message passing algorithm that applies light-weight inference to the same kind of graphical model. TableMiner is similar in the way that the iterative *backward-update* phase could also be considered a semantic message passing process that involves fewer variables and factors, hence is faster to converge.

One limitation of the above methods is that the construction of candidate space and their feature representation is *exhaustive*, since they require evidence from all content cells of a column in order to classify that column. This can significantly damage the efficiency of semantic Table Interpretation algorithms as it is shown that constructing candidate space and their feature representations is the major bottleneck in Table Interpretation [12]. However, as illustrated before, human cognition does not necessarily follow the similar process but can be more efficient as we are able to infer on partial data.

Another issue with existing work is that many of them make use of non-generalizable, knowledge base specific features. For example, Venetis et al. [22] and Wang et al. [23] use statistics gathered during the construction of the knowledge bases, which is unavailable in resources such as Freebase[3] or DBpedia[4]. Syed et al. [21] and Mulwad et al. [17, 15, 16] use search relevance scores returned by the knowledge base that is also

[3] http://www.freebase.com/
[4] http://dbpedia.org/

resource-specific and unavailable in, e.g., Freebase and DBpedia. TableMiner however, uses only generic features present in almost every knowledge base.

2.2 Information Extraction in general

The three subtasks tackled by semantic Table Interpretation are closely related to Named Entity Recognition (NER), Named Entity Disambiguation and Relation Extraction in the general Information Extraction domain. State-of-the-art methods [20, 10] however, are tailored to unstructured text content that is different from tabular data. The inter-dependency among the table components cannot be easily taken into account in such methods [14]. For NER and Relation Extraction, a learning process is typically required for each semantic label (i.e., class or relation) that must be known a-priori and training or seed data must be provided. In Table Interpretation however, semantic classes and relations are unknown a-priori. Further, due to the large candidate space, it is infeasible to create sufficient training or seed data in such tasks.

Wrapper induction [11, 8] automatically learns wrappers that can extract information from structured Webpages. It builds on the phenomenon that the same type of information are typically presented in similar structures in different Webpages and exploits such regularities to extract information. Technically, Wrapper induction can be adapted to partially address Table Interpretation by learning wrappers able to classify table columns. However, the candidate classes must be defined a-priori and training data are essential to build such wrappers. As discussed above, these are infeasible in the case of semantic Table Interpretation.

3 Methodology

This section describes TableMiner in details. T denotes a regular, horizontal, relational table containing i rows of content cells (excluding the row of table headers) and j columns, T_i denotes row i, T_j denotes column j, TH_j is the header of column j, and $T_{i,j}$ is a cell at row T_i and column T_j. X denotes different types of *context* used to support Table Interpretation. C_j is a set of candidate concepts for column j. $E_{i,j}$ is a set of candidate entities for the cell $T_{i,j}$. Both C_j and $E_{i,j}$ are derived from a reference knowledge base, details of which is to be described below. Function $l(o)$ returns the string content if o is a table component (e.g., in Figure 1 $l(T_{2,1})$ ='Yliki'), or the label if o is an annotation (i.e., any $c_j \in C_j$ or any $e_{i,j} \in E_{i,j}$). Unless otherwise stated, $bow(o)$ returns a bag-of-words (multiset) representation of o by tokenizing $l(o)$, then normalizing each token by lemmatization and removing stop words. $bowset(o)$ is the de-duplicated set based on $bow(o)$. w is a single token and $freq(w, o)$ counts the frequency of w in $bow(o)$. $|\cdot|$ returns the size of a collection, either containing duplicates or de-duplicated.

TableMiner firstly identifies table columns that contain mostly ($> 50\%$ of non-empty rows) named entities (NE-columns). This is done by using regular expressions based on capitalization and number of tokens in each content cell. The goal is to distinguish such columns from others that are unlikely to contain named entities (e.g., columns containing numeric data, such as column 'Area' in Figure 1 and thus do not

Context	
Webpage title	out-table context
Table caption	
Semantic markups if any	
Surrounding paragraphs	
Column header	in-table context
Row content	
Column content	

Table 1. Table context elements

need classification. Then given an NE-column T_j, **column interpretation** proceeds in a two-phase bootstrapping manner, where each phase deals with both column classification and cell entity disambiguation. The first *forward-learning* phase builds initial interpretation based on partial data in the column, while the second *backward-update* phase interprets remaining cells and iteratively updates annotations (concept and entity) for the entire column until they are stablized.

3.1 Context

A list of the context types used for semantic Table Interpretation is shown in Table 1. A key innovation in TableMiner is using context *outside* tables, including **table captions**, **Webpage title**, **surrounding paragraphs**, and **semantic markups** inserted by certain websites.

Table captions and the title of the Webpage may mention key terms that are likely to be the focus concept in a table. Paragraphs surrounding tables may describe the content in the table, thus containing clue words indicating the concepts or descriptions of entities in the table. Furthermore, an increasing number of semantically annotated Webpages are becoming available under the heavily promoted usage of semantic markup vocabularies (e.g., microdata format at *schema.org*) by major search engines [6]. An example of this is IMDB.com, on which Webpages about movies contain microdata annotations such as movie titles, release year, directors and actors, which are currently used by Google Rich Snippet[5] to improve content access. Such data provides important clues on the 'aboutness' of a Webpage, and therefore tables (if any) within the Webpage.

3.2 The *forward-learning* phase

Algorithm 1 shows the *incremental inference with stopping* (*I-inf*) algorithm used by *forward-learning*. Each iteration disambiguates a content cell $T_{i,j}$ by comparing candidate entities from $E_{i,j}$ against their context and choosing the highest scoring (i.e., winning) candidate (**Candidate search** and **Disambiguation**). Then the concepts associated with the entity are gathered to create C_j the set of candidate concepts for column T_j, and each member $c_j \in C_j$ is scored based on its context and those already disambiguated entities (**Classification**). At the end of each iteration, C_j from the current

[5] http://www.google.com/webmasters/tools/richsnippets

iteration is compared with the previous to check for convergence, by which 'satisfactory' initial annotations are created and *forward-learning* ends.

Algorithm 1 Forward learning

```
Input: T_j; C_j ← ∅
for all cell T_{i,j} in T_j do
    prevC_j ← C_j
    E_{i,j} ←disambiguate(T_{i,j})
    C_j ←updateclass(C_j, E_{i,j})
    if convergence(C_j, prevC_j) then
        break
    end if
end for
```

Candidate search In this step, the text content of a cell $l(T_{i,j})$ is searched in a knowledge base and entities whose labels $l(e_{i,j})$ overlaps with $l(T_{i,j})$ is chosen as candidates ($E_{i,j}$) for the cell (the number of overlapping words/tokens does not matter). For example, 'Trichonida' will retrieve candidate named entities 'Lake Trichonida' and 'Trichonida Province'. TableMiner does not use relevance-based rankings or scores returned by the knowledge base as features for inference, while others [21, 17, 15, 16] do.

Disambiguation ($disambiguate(T_{i,j})$) Each content cell $T_{i,j}$ is disambiguated by candidate entity's confidence score, which is based on two components: a *context* score $ctxe$ and a *name match* score nm.

The *context* score measures the similarity between each candidate entity and the context of $T_{i,j}$, denoted as $x_{i,j} \in X_{i,j}$. Firstly, a $bow(e_{i,j})$ representation for each $e_{i,j} \in E_{i,j}$ is created based on triples containing $e_{i,j}$ as subject. Let $<e_{i,j}, predicate, object>$ be the set of such triples retrieved from a knowledge base, then $bow(e_{i,j})$ simply concatenates $object$ from all triples, tokenizes the concatenated string, and normalizes the tokens by lemmatization and stop words removal. For each $x_{i,j}$, $bow(x_{i,j})$ converts the text content of the corresponding component into a bag-of-words representation following the standard definition introduced before. Finally, to compute the similarity between $e_{i,j}$ and $x_{i,j}$, two functions are used. For each type of out-table context shown in Table 1, the similarity is computed using a frequency weighted dice function:

$$dice(e_{i,j}, x_{i,j}) = \frac{2 \times \sum_{w \in bowset(e_{i,j}) \cap bowset(x_{i,j})} (freq(w, e_{i,j}) + freq(w, x_{i,j}))}{|bow(e_{i,j})| + |bow(x_{i,j})|} \quad (1)$$

For in-table context, the similarity is computed by 'coverage':

$$coverage(e_{i,j}, x_{i,j}) = \frac{\sum_{w \in bowset(e_{i,j}) \cap bowset(x_{i,j})} freq(w, x_{i,j})}{|bow(x_{i,j})|} \quad (2)$$

, because the sizes of $bow(e_{i,j})$ and $bow(x_{i,j})$ can be often different orders of magnitude and Equation 1 may produce negligible values. Specifically, *row content* is the concatenation of $l(T_{i,j'})$ for all columns $j', j \neq j'$ (e.g., for the cell 'Yliki' in Figure 1 this includes 'Boeotia','22,731'). Intuitively, these are likely to be attribute data of the concerning entity. *Column content* is the concatenation of $l(T_{i',j})$ for all rows $i', i \neq i'$ (e.g., for the cell 'Yliki' in Figure 1 this includes 'Trichonida','Amvrakia', and 'Lysimachia'). Intuitively, these are names of entities that are semantically similar.

Therefore, the similarity score between $e_{i,j}$ and each $x_{i,j} \in X_{i,j}$ is computed as above and summed up to obtain the context score $ctxe(e_{i,j})$.

The *name match* score examines the overlap between the name of the entity and the cell content, to promote entities whose name matches exactly the content string:

$$nm(e_{i,j}, T_{i,j}) = \sqrt{\frac{2 \times |bowset(l(e_{i,j})) \cap bowset(T_{i,j})|}{|bowset(l(e_{i,j}))| + |bowset(T_{i,j})|}} \quad (3)$$

The final confidence score of a candidate entity, denoted by $fse(e_{i,j})$, is the product of $ctxe(e_{i,j})$ and $nm(e_{i,j}, T_{i,j})$.

Classification ($updateclass(C_j)$) In each iteration, the entity with the highest $fse(e_{i,j})$ score is selected for the current cell and its associated concepts are used to update the candidate set of concepts C_j for the column. Each $c_j \in C_j$ is associated with a confidence score $fsc(c_j)$ also consisting of two elements: a *base* score bs and a *context* score $ctxc$.

The *base* score is based on the *fse* scores of the winning entities from already disambiguated content cells by the current iteration. Let $disamb(c_j)$ be the sum of $fse(e_{i,j})$ where $e_{i,j}$ is a winning entity from a content cell and is associated with c_j, then $bs(c_j)$ is $disamb(c_j)$ divided by the number of rows in T. Note that as additional content cells are disambiguated in new iterations, new candidate concepts may be added to C_j; or for existing candidate concepts, their base scores can be updated if the winning entities from newly disambiguated content cell also select them.

The *context* score is based on the overlap between $l(c_j)$ and its context, and is computed in the similar way as the context score for candidate entities. Let $x_j \in X_j$ denotes various types of context for the column header TH_j. All types of context shown in Table 1 except row content is used. For each context $x_j \in X_j$, a similarity score is computed between c_j and x_j using the weighted dice function introduced before but replacing $e_{i,j}$ with c_j, and $x_{i,j}$ with x_j. $bow(c_j)$ and $bowset(c_j)$ is created following the standard definitions. Then the *sum* of the similarity scores becomes the context score $ctxc(c_j)$.

The final confidence score of a candidate concept $fsc(c_j)$ adds up $bs(c_j)$ and $ctxc(c_j)$ with equal weights.

Convergence (convergence(C_j, $prevC_j$)) Results of the two above operations at each iteration may either create new concept candidates for the column, or resetting the scores of existing candidates, thus changing the 'state' of C_j. TableMiner does not exhaustively process every cell in a column. Instead, it automatically stops by detecting the convergence of 'entropy' of the state of C_j at the end of an iteration as measured below. Convergence happens if the difference between the current and previous state's entropy is less than a threshold of t.

$$entropy(C_j) = -\sum_{c_j \in C_j} P(c_j) \log_2 P(c_j) \tag{4}$$

$$P(c_j) = \frac{fsc(c_j)}{\sum_{c'_j \in C_j} fsc(c'_j)} \tag{5}$$

The intuition is that when the entropy level stabilizes, the contribution by each $P(c_j)$ to the state is also expected to stabilize. In other words, the relative confidence score of c_j to the collective sum (the denominator in Equation 5) changes little. As a result, the ranking of candidate concepts also stabilizes, and so winning candidates will surface.

3.3 The *backward-update* phase

The *backward-update* phase begins (i.e., first iteration) by taking the classification outcome C_j from the *forward* phase as constraints on the disambiguation of remaining cells in the same column. Let $C_j^+ \subset C_j$ be the set of highest scoring classes ('winning' concepts, multiple concepts with the same highest score is possible) for column j computed by the *forward* phase. For each remaining cell in the column, disambiguation candidates are restricted to entities whose associated concepts overlap with C_j^+. Effectively, this reduces the number of candidates thus improving efficiency. **Disambiguation** follows the same procedure as in the *forward* phase, and its results may revise **classification** C_j for the column, either adding new elements to C_j, or resetting scores of existing ones (due to changes of $fsc(c_j)$).

Thus after disambiguating the remaining cells, C_j^+ is re-selected. If the new C_j^+ is different from the previous, a new update operation is triggered. It repeats the disambiguation and classification operations on the entire column, while using the new C_j^+ as constraints to restrict candidate entity space. This procedure repeats until C_j^+ and the winning entity in each cell stabilizes (i.e., no change), completing interpretation.

In theory, starting from the second iteration, new candidate entities may be retrieved and processed due to the change in C_j^+. Empirically, it is found that 1) in most cases the update phase completes in one iteration; and 2) in cases where it doesn't, it converges fast and following iterations mostly re-selects from the pool of candidates that were already processed in the beginning of the update phase (first iteration), thus incurring little computational cost.

4 Evaluation

TableMiner is evaluated by the standard Precision, Recall and F1 metrics in the column classification and entity disambiguation tasks. It is compared against two baselines on two datasets (shown in Table 2). The knowledge base used in this experiment is Freebase. Freebase is currently the largest well-maintained knowledge base in the world, containing over 2.4 billion facts about over 43 million topics (e.g., entities, concepts), largely exceeding other popular knowledge bases such as DBpedia and YAGO.

4.1 Datasets

Limaye112 contains a randomly selected 112 tables from the Limaye dataset [12]. The original dataset is annotated by Wikipedia article titles referring to named entities and YAGO concepts and relations. The dataset contains about 90% of Wikipedia article pages, while the other 10% are randomly crawled Webpages. Each Webpage contains a 'focus' relational table to be interpreted, together with the context such as page titles, table captions, and paragraphs around it. These Webpages do not have Microdata annotations. The dataset covers multiple domains, such as film, music, games, location, organization, events etc. These tables must be re-annotated due to the significant changes of such resources, also due to the usage of a different knowledge base in this work.

To create the ground truth for the classification task, the NE-columns in these tables are manually annotated following a similar process as Venetis et al. [22]. Specifically, TableMiner and the baselines (Section 4.2) are ran on these tables and the candidate concepts for all NE-columns are collected and presented to annotators. The annotators mark each label as *best*, *okay*, or *incorrect*. The basic principle is to prefer the most specific concept among all suitable candidates. For example, given a content cell 'Penrith Panthers', the concept 'Rugby Club' is the *best* candidate to label its parent column while 'Sports Team' and 'Organization' are *okay*. The annotators may also insert new labels if none of the candidates are suitable.

The top ranked prediction by TableMiner is checked against the classification ground truth. Each *best* label is awarded a score of 1 while each *okay* label is awarded 0.5. Further, if there are multiple top-ranked candidates, each candidate considered correct only receives a fraction of its score as $\frac{score}{\#topranked}$. For example, if a column containing film titles has two top-ranked concept candidates with the same score: 'Film' (best) and 'Book' (incorrect), this prediction receives a score of 0.5 instead of 1. This is to penalize the situation where the Table Interpretation system fails to discriminate false positives from true positives.

To create the ground truth for the disambiguation task, each Wikipedia article title in the original tables is automatically mapped to a Freebase topic (e.g., an entity or a concept) id by using the MediaWiki API[6] and the Freebase MQL[7] interface. As it will be discussed later, evaluation of entity disambiguation on this dataset reveals that the original dataset could be biased, possibly due to the older version of Wikipedia used in

[6] http://www.mediawiki.org/wiki/API:Main_page
[7] http://www.freebase.com/query

	Limaye112	IMDB
Tables	112	7,354
Annotated columns	254 (119)	7,354
Annotated entity cells	2,089	92,317

Table 2. Datasets for evaluation. The number in bracket shows the number of annotated columns for the corresponding 112 tables in the original Limaye dataset. The re-created dataset doubles the size of annotations.

the original experiments. Therefore, a manual inspection of the output of TableMiner and the baselines evaluation is carried out to further evaluate the different systems.

IMDB contains 7,354 tables extracted from a random set of IMDB movie Webpages. They are annotated automatically to evaluate entity disambiguation. Each IMDB movie Webpage[8] contains a table listing a column of actors/actresses and a column of corresponding characters played. Cells in the actor/actress column are linked with an IMDB item ID, which, when searched in Freebase, returns a unique (if any) mapped Freebase topic. Thus these columns are annotated automatically in such a way. The 'character' column is not used since they are not mapped in Freebase.

4.2 Configuration and baseline

The convergence threshold in the *I-inf* algorithm is set to 0.01. Semantic markups are only available in the IMDB dataset. To use this type of context for semantic Table Interpretation, Any23[9] is used to extract the microdata format annotations as RDF triples and the *objects* of triples are concatenated as contextual text. Annotations within the HTML <*table*> tags are excluded.

Two **baselines** are created. Baseline 'first result' (B_{first}) firstly disambiguates every content cell in a column by choosing the top ranked named entity candidate in the Freebase search result. Freebase implements a ranking algorithm for its Search API to promote popular topics. TableMiner however, does not use such features. Then each disambiguated cell casts a vote to the set of concepts the winning named entity belongs to, and the concept that receives the majority vote is selected to label the column.

Baseline 'similarity based' (B_{sim}) uses both string similarity methods and a simple context-based similarity measure to disambiguate a content cell. Given a content cell and its candidate named entities, it computes a string similarity score between a candidate entity's name and the cell content using the Levenshtein metric. It then uses Equation 1 to compute a context overlap score between the bag-of-words representation of a candidate entity and the row context of its containing cell in the table. The two scores are added together as the final disambiguation score for a candidate named entity for the cell and the winning candidate is chosen for the cell. Candidate concepts for the column are derived from winning named entity for each content cell, then the score of a candidate concept is based on the fraction of cells that cast vote for that concept, plus the string similarity (Levenshtein) between the label of the concept and the column

[8] e.g., http://www.imdb.com/title/tt0071562/

[9] https://any23.apache.org/

	B_{first}	B_{sim}	TableMiner
In-table context	No	Yes	Yes
Out-table context	No	No	Yes

Table 3. Use of context features in the three methods for comparison

	Limaye112			IMDB		
	Precision	Recall	F1	Precision	Recall	F1
B_{first}	0.927	0.918	**0.922**	0.927	0.922	0.925
B_{sim}	0.907	0.898	0.902	0.937	0.932	0.935
TableMiner	0.923	0.921	**0.922**	0.96	0.954	**0.956**

Table 4. Disambiguation results on the two datasets. The highest F1 on each dataset is marked in **bold**.

header text. Baseline B_{sim} can be considered as an 'exhaustive' Table Interpretation method, which disambiguates every content cell before deriving column classification and uses features from in-table context that are commonly found in state-of-the-art [12, 21, 17, 15, 16].

Table 3 compares the three methods in terms of the contextual features used for learning.

4.3 Results and discussion

Effectiveness Table 4 shows disambiguation results obtained on the two datasets. TableMiner obtains the best F1 on both datasets. It also obtains the highest Precision and Recall on the IMDB dataset, and the highest Recall on the Limaye112 dataset. It is surprising to note that even the most simplistic baseline B_{first} obtains very good results: it achieves over 0.9 F1 on the IMDB dataset, while on the Limaye112 dataset it obtains results that betters B_{sim} and equally compares to TableMiner. Note that the figures are significantly higher than those reported originally by Limaye et al (in the range of 0.8 and 0.85) [12].

The extremely well performance on the IMDB dataset could be attributed to the domain and the Freebase search API. As mentioned before, the Freebase search API assigns higher weights to popular topics. The result is that topics in the domains such as movie, book, pop music and politics are likely to be visited and edited more frequently, subsequently increasing their level of 'popularity'. Therefore, by selecting the top-ranked result, B_{first} is very likely to make the correct prediction.

To uncover the contributing factors to its performance on the Limaye112 dataset, the ground truth is analyzed and it is found that, each of the 112 tables has on average only 1.1 (minimum 1, maximum 2) columns that are annotated with entities, while TableMiner annotates on average 2.3 NE-columns. This suggests that the entity annotations in the ground truth are sparse. Moreover, the average length of entity names by number of tokens is 2.3, with the maximum being 12, and over 33% of entity names have 3 or more tokens while only 22% have a single-token name. This could possibly explain the extremely well performance by B_{first} as typically, short names are much

	Precision	Precision either-or
B_{first}	0.265	0.431
B_{sim}	0.306	0.498
TableMiner	0.491	0.801

Table 5. Precision based on manual analysis of 932 entity annotations that the three systems disagree on.

more ambiguous than longer names. By using a dataset that is biased toward entities with long names, the strategy by B_{first} is very likely to succeed.

Hence to obtain a more balanced perspective, the results created by the three systems are manually inspected and re-annotated. To do so, for each method, the predicted entity annotations that are already covered by the automatically created ground truth are excluded. Then, in the remaining annotations, those that all three systems predict the same are removed. The remainder (932 entity annotations, of which 572 is predicted correctly by at least one system) are the ones that the three systems 'disagree' on, and are manually validated. Table 5 shows the analysis results. TableMiner significantly outperforms the two baseline. Manual inspection on 20% of the wrong annotations by all three methods reveals that it is largely ($> 80\%$) because the knowledge base does not contain the correct candidate. When only annotations that are correct by any one method are considered (Precision either-or), TableMiner achieves a precision of 0.8 while B_{first} 0.431 and B_{sim} 0.498.

Table 6 shows the classification result on the Limaye112 dataset. TableMiner almost tripled the performance of B_{first} and significantly outperforms B_{sim}. Again surprisingly, the superior performance by B_{first} on the disambiguation task does not translate to equal performance on the classification task. Manual inspection shows that it is significantly penalized by predicting multiple top-ranked candidate concepts. In other words, it fails to discriminate true positives from false positives. It has been noted that the state-of-the-art methods often use a concept hierarchy defined within knowledge bases to solve such cases by giving higher weights to more specific concepts [12, 16]. However, concept hierarchies are not necessarily available in all knowledge bases. For example, Freebase has a rather loose concept network instead of a hierarchy. Nevertheless, TableMiner is able to predict a single best concept candidate in most cases without such knowledge. It also outperforms B_{sim} by a substantial margin, suggesting the usage of various table context is very effective and that exhaustive approaches may not necessarily offer advantage but causes additional computation.

While not directly comparable due to the datasets and knowledge bases used, the classification results by TableMiner is higher than 0.56 in [12], 0.6-0.65 (*best* or *okay*) in [22], and 0.5-0.6 (*best* or *okay*) in [16].

Efficiency The potential efficiency improvement by TableMiner can be assessed by observing the reduced number of candidate entities. As discussed earlier, candidate retrieval and feature space construction account the majority (>90%) of computation [12]. Experiments in this work show that retrieving candidate entities and their data

	best only			*best* or *ok*		
	Precision	Recall	F1	Precision	Recall	F1
B_{first}	0.258	0.247	0.252	0.505	0.484	0.494
B_{sim}	0.487	0.481	0.484	0.667	0.658	0.662
TableMiner	0.646	0.618	**0.632**	0.761	0.729	**0.745**

Table 6. Classification result on the Limaye112 dataset. The highest F1 is marked in **bold**.

(triples) from Freebase accounts for over 90% of CPU time, indeed a major bottleneck in semantic Table Interpretation.

TableMiner reduces the quantity of candidate entities to be processed by 1) generating initial interpretation (*forward-learning*) using *partial* instead of *complete* data as an exhaustive method would otherwise do; and 2) using the initial interpretation outcome to constrain further learning (*backward-update*). Compared against B_{sim} that exhaustively disambiguates every content cells in a table before classifying the columns, TableMiner reduces the total number of candidate entities to be considered by disambiguation operations by 32% in the Limaye112 and 24% in the IMDB datasets respectively. When only content cells processed in the *backward-update* phase are considered, the figures amount to 38% and 61%.

Furthermore, Table 7 shows the convergence speed in the *forward-learning* phase. Considering the column classification task only and using the Limaye112 dataset as an example, it suggests that on average only 9 rows are needed to create *initial* classification labels on NE-columns, as opposed to using all rows in a table (average of 27) by an exhaustive method. The slowest convergence happens in a table of 78 rows (converged at 46). The average fractions of rows that need not to be processed (i.e., savings) in the *forward* phase are 57% for Limaye112 and 43% for IMDB. Then In the *backward* phase, the number of columns that actually need iterative update is 10% of all columns in both datasets. The average number of iterations for those needing iterative update is 3 for both datasets. To summarize again using the Limaye112 dataset, TableMiner manages to produce 'stable' column classification for 90% of columns using only *forward-learning* with an average of 9 rows, resulting in a potential 57% of savings than an exhaustive method for this very specific task.

Final remark In summary, by using various types of in- and out-table context in semantic Table Interpretation, TableMiner obtains the best performance on both classification and disambiguation tasks. On the classification task, it delivers significant improvement by between 8 and 38% over the baselines, none of which uses features from out-table context. By adopting an incremental, bootstrapping pattern of learning, TableMiner can potentially deliver between 24 and 60% computational savings compared against exhaustive methods depending on tasks.

5 Conclusion

This paper introduced TableMiner, a Table Interpretation method that makes use of various types of context both within and outside tables for classifying table columns and

	Max	Min	Mean
Limaye112	42	2	9
IMDB	15	2	8

Table 7. Number of iterations until convergence in the *forward learning* phase.

disambiguating content cells, and learns in an incremental, bootstrapping, and mutually-recursive pattern. TableMiner contributes to the state-of-the-art by introducing 1) a generic Table Interpretation method able to be adapted to any knowledge bases; 2) a generic model of using various table context in such task, the first that uses semantic markups within Webpages as features; 3) an automatic method for determining sample data to bootstrap Table Interpretation.

TableMiner is evaluated on two datasets against two baselines, one of which represents an exhaustive method that only uses features from within-table context. TableMiner consistently obtains the best results on both tasks. It significantly outperforms both baselines in the classification task and on the re-annotated dataset (i.e., manual inspection and validation) in the disambiguation task.

One limitation of the current work is that the contribution of each type of out-table and in-table context in the task is not extensively evaluated. This will be addressed in future work. Further, future work will also focus on extending TableMiner to a full Table Interpretation method addressing all three subtasks. Other methods of sample selection will also be explored and compared.

Acknowledgement Part of this research has been sponsored by the EPSRC funded project LODIE: Linked Open Data for Information Extraction, EP/J019488/1

References

1. Adelfio, M.D., Samet, H.: Schema extraction for tabular data on the web. Proc. VLDB Endow. 6(6), 421–432 (Apr 2013)
2. Ahmad, A., Eldad, L., Aline, S., Corentin, F., Raphaël, T., David, T.: Improving schema matching with linked data. In: First International Workshop On Open Data (2012)
3. Bhagavatula, C.S., Noraset, T., Downey, D.: Methods for exploring and mining tables on wikipedia. In: Proceedings of the ACM SIGKDD Interative Data Exploration and Analysis (IDEA). IDEA'13 (2013)
4. Cafarella, M.J., Halevy, A., Madhavan, J.: Structured data on the web. Communications of the ACM 54(2), 72–79 (Feb 2011)
5. Cafarella, M.J., Halevy, A., Wang, D.Z., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. Proceedings of VLDB Endowment 1(1), 538–549 (Aug 2008)
6. Ciravegna, F., Gentile, A.L., Zhang, Z.: Lodie: Linked open data for web-scale information extraction. In: Maynard, D., van Erp, M., Davis, B. (eds.) SWAIE. CEUR Workshop Proceedings, vol. 925, pp. 11–22. CEUR-WS.org (2012)
7. Cucerzan, S.: Large-scale named entity disambiguation based on Wikipedia data. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). pp. 708–716. Association for Computational Linguistics, Prague, Czech Republic (June 2007)

8. Gentile, A.L., Zhang, Z., Augenstein, I., Ciravegna, F.: Unsupervised wrapper induction using linked data. In: Proceedings of the seventh international conference on Knowledge capture. K-CAP '13, ACM, New York, NY, USA (2013)
9. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th Conference on Computational Linguistics - Volume 2. pp. 539–545. COLING '92, Association for Computational Linguistics, Stroudsburg, PA, USA (1992)
10. Krishnan, V., Manning, C.D.: An effective two-stage model for exploiting non-local dependencies in named entity recognition. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics. pp. 1121–1128. ACL-44, Association for Computational Linguistics, Stroudsburg, PA, USA (2006)
11. Kushmerick, N., Weld, D.S., Doorenbos, R.: Wrapper induction for information extraction. In: Proc. IJCAI-97 (1997)
12. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. Proceedings of the VLDB Endowment 3(1-2), 1338–1347 (2010)
13. Ling, X., Halevy, A., Wu, F., Yu, C.: Synthesizing union tables from the web. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence. pp. 2677–2683 (2013)
14. Lu, C., Bing, L., Lam, W., Chan, K., Gu, Y.: Web entity detection for semi-structured text data records with unlabeled data. International Journal of Computational Linguistics and Applications (2013)
15. Mulwad, V., Finin, T., Joshi, A.: Automatically generating government linked data from tables. In: Working notes of AAAI Fall Symposium on Open Government Knowledge: AI Opportunities and Challenges (November 2011)
16. Mulwad, V., Finin, T., Joshi, A.: Semantic message passing for generating linked data from tables. In: International Semantic Web Conference (1). pp. 363–378. Lecture Notes in Computer Science, Springer (2013)
17. Mulwad, V., Finin, T., Syed, Z., Joshi, A.: T2ld: Interpreting and representing tables as linked data. In: Polleres, A., Chen, H. (eds.) ISWC Posters and Demos. CEUR Workshop Proceedings, CEUR-WS.org (2010)
18. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Linguisticae Investigationes 30(1), 3–26 (January 2007), publisher: John Benjamins Publishing Company
19. Sarawagi, S.: Information extraction. Found. Trends databases 1(3), 261–377 (Mar 2008)
20. Sarawagi, S., Cohen, W.W.: Semi-markov conditional random fields for information extraction. In: In Advances in Neural Information Processing Systems 17. pp. 1185–1192 (2004)
21. Syed, Z., Finin, T., Mulwad, V., Joshi, A.: Exploiting a web of semantic data for interpreting tables. In: Proceedings of the Second Web Science Conference (April 2010)
22. Venetis, P., Halevy, A., Madhavan, J., Paşca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. Proceedings of VLDB Endowment 4(9), 528–538 (Jun 2011)
23. Wang, J., Wang, H., Wang, Z., Zhu, K.Q.: Understanding tables on the web. In: Proceedings of the 31st international conference on Conceptual Modeling. pp. 141–155. ER'12, Springer-Verlag, Berlin, Heidelberg (2012)
24. Wu, W., Li, H., Wang, H., Zhu, K.Q.: Probase: a probabilistic taxonomy for text understanding. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 481–492. SIGMOD '12, ACM, New York, NY, USA (2012)
25. Zanibbi, R., Blostein, D., Cordy, J.: A survey of table recognition: Models, observations, transformations, and inferences. International Journal of Document Analysis and Recognition 7, 1–16 (2003)