Undefined 1 (2009) 1–5
IOS Press

Start Small, Build Complete: Effective and Efficient Semantic Table Interpretation using TableMiner

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Ziqi Zhang *
Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello, Sheffield, S1 4DP
E-mail: z.zhang@dcs.shef.ac.uk

Abstract. This article introduces TableMiner, the first semantic Table Interpretation method able to annotate Web tables using an incremental, bootstrapping learning approach seeded by automatically selected 'partial' content from tables. The basic principle is to create initial and partial annotations of a table using some as opposed to all content in the table. The partial outcome then serves as 'stepping stones' to guide interpretation of remaining content in the table, followed by a process of iteratively refining results on the entire table to create final optimal annotations. To construct feature representations, TableMiner uses various types of contextual information both inside and outside tables, including pre-defined semantic markups (e.g., RDFa/Microdata annotations) within some webpages that to the best of the author's knowledge, have never been used in Natural Language Processing tasks. Evaluated on the largest collection of datasets known to-date including four datasets with a total of more than 15,000 tables, TableMiner consistently outperforms four baselines under all experimental settings. On the two most representative datasets covering multiple domains and various table schemata, it achieves improvement in F1 by between 1 and 42 percentage points depending on specific tasks. Compared against state-of-the-art, it also obtains higher results on similar datasets. The bootstrapping learning strategy seeded by partial table content also enables TableMiner to be very efficient. Empirically, it reduces data to be processed by up to 66% or 29% savings in CPU time when compared against classic methods that 'exhaustively' processes the entire table content to build features for interpretation.

Keywords: Web table, Named Entity Recognition, Named Entity Disambiguation, Relation Extraction, Linked Data, semantic Table Interpretation, table annotation

1. Introduction

Recovering semantics from tables is becoming crucial 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 [2,3]. On the other hand, search engines typically ignore underlying semantics of such structures at indexing, hence performing poorly on tabular data [19,23]. While substantial effort has focused on converting tabular data from legacy structured data sources such as databases and spreadsheets into linked data [11,18,26,6], these methods fail to uncover the richer, implicit semantics of tables [23]; also the methods are bound with specific data structures and cannot be directly used on Web tables. More recent research directed to this problem is semantic **Table Interpretation** [13,14,19,24,29,22,30,31,1,23,21] that aims to tackle three tasks in tables. Given a well-formed *relational table* (e.g., Figure 1) and reference sets of con-

*Corresponding author. E-mail: z.zhang@dcs.shef.ac.uk.

cepts, named entities and relations, (1) disambiguate named entity mentions in content cells by linking them to existing reference set of named entities (if any); (2) label columns with semantic concepts if they contain named entity mentions (**NE-columns**), or properties of concepts if they contain data literals (**literal-columns**); and (3) identify the semantic relations between columns. The outcome of this process is semantically annotated tabular data, which does not only enable effective indexing and search of the data, but ultimately can be transformed to create new triples (e.g., new instances of concepts and relations) to populate the Linked Open Data (LOD) cloud.

While these closely relate to classic Natural Language Processing (NLP) and Information Extraction (IE) research problems [38,9,5,25], it is known that traditional NLP methods are typically trained for well-formed sentences in unstructured texts, therefore, are unlikely to succeed on tabular data due to different content representations in tables [19,23]. Instead, typical Table Interpretation methods make extensive use of rich lexical and semantic information from **knowledge bases** that define a set of concepts, named entities and relations. In the generic form, these resources are interlinked as triples and hence the knowledge bases are essentially linked data sets. The general workflow involves (1) retrieving candidates corresponding to table components (e.g., candidate concepts given a column header, or candidate named entities given the text of a content cell) from the knowledge base, (2) constructing features of candidates to model semantic interdependence between table components and candidates (e.g., the header text of a column and the name of a candidate concept), and between various table components (e.g., a column header is expected to describe a concept shared by all named entities in the content cells from the column), and (3) applying inference to choose the best candidates.

Despite an increasing effort on this research, existing methods suffer from several limitations, which motivated this work. To illustrate, consider Figure 1 that shows part of a webpage with a table containing over 60 rows. To interpret this table, state-of-the-art methods would make inference based on the entire table content. However, from a human reader point of view, this is unnecessary. For example, simply reading the the eight rows shown in Figure 1 one can confidently assign a concept to each column to best describe its content and infer that the first column forms binary relations with other columns in the table. Being able to make such inference with limited data would give substantial efficiency advantage to Table Interpretation algorithms since the first two steps of Table Interpretation usually take the majority of computation [19].

Further, existing methods predominantly use two types of features: those derived from background knowledge bases (i.e., triples from certain linked data sets) and those derived from table components (*in-table* context) such as header text, and row content. This work notes that the context around and outside tables can also provide useful clues to the task. For example, the word 'film' is repeated 17 times on the webpage, one of which is within the table title 'Notable films'. It is a strong hint of suitable concepts to label the first column, whose header 'Title' is a less indicative feature in this case. Such *out-table* features can be defined in different ways to support interpretation. In particular, another source of linked data - the pre-defined semantic markups within webpages such as RDFa/Microdata annotations - provide high-quality, 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.

Additionally, tables consist of both NE-columns containing named entity mentions, and literal-columns containing data values of named entities on the corresponding rows (E.g., column 'Year'). However, many existing methods only deal with NE-columns.

To address these issues, this article introduces **TableMiner**, a semantic Table Interpretation method built on the principle of 'start small, build complete'. That is, (1) build initial, likely erroneous interpretation based on partial table content and simple models assuming limited interdependence between table components; (2) iteratively optimize the initial interpretation by enforcing interdependence between table components; and (3) tackle the three tasks incrementally, one built on another. Concretely, TableMiner firstly interprets NE-columns (to be called *column interpretation*), while coupling column classification and named entity disambiguation in a bootstrapping pattern that consists of a *LEARNING* phase and an *UPDATE* phase. The *LEARNING* phase interprets each column independently, by firstly learning to create initial column labels (column classification) using an automatically determined *sample* of the column, followed by *constrained* (limiting candidate space) named entity disambiguation in the column. The *UPDATE* phase iteratively optimizes the classification and disambiguation results in each column based on a notion of 'domain consensus' that captures inter-column dependency, creating

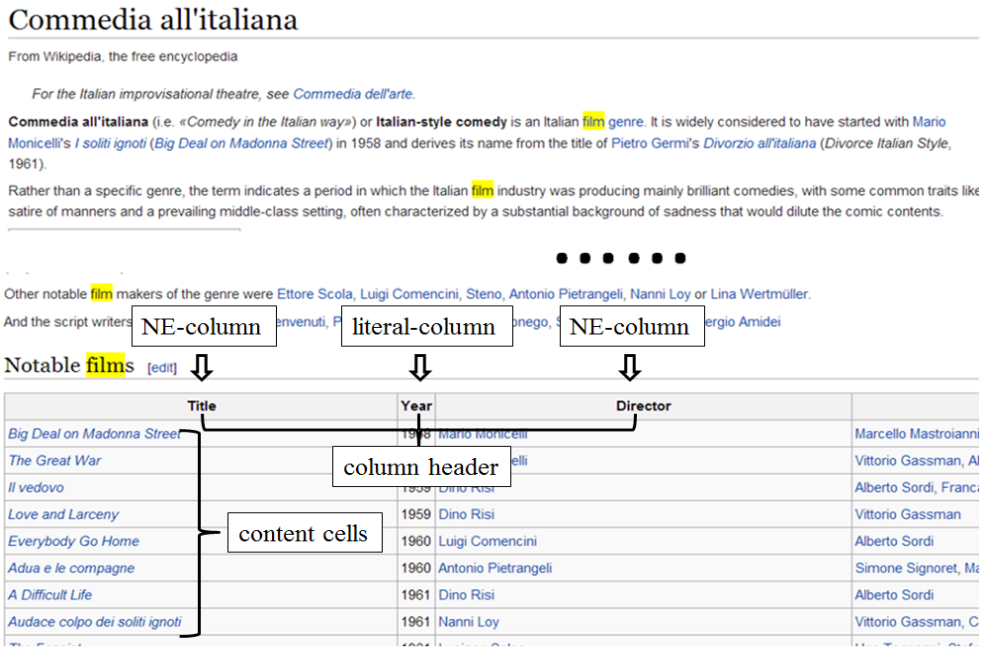

Fig. 1. An example Wikipedia webpage containing a relational table (Last retrieved on 9 April 2014).

a global optimum. For relation enumeration, TableMiner asserts a 'subject' column [30,31] in a table and infers its relations with other - both NE- and literal-columns in the table based on column interpretation results. In all three tasks, TableMiner defines and uses various features from both in- and out-table context.

TableMiner is evaluated on four datasets with a total of over 15,000 tables, the largest collection of datasets for this task known to date. TableMiner consistently obtains the best performance on all datasets in any subtask. On the two most representative datasets covering multiple domains and diverse table schemata, it obtains an improvement of about 1-4 percentage points in disambiguation, 6-42 in classification, and 4-11 in relation enumeration. It also obtains higher figures than those reported by state-of-the-art on similar datasets. Analysis shows that it is very efficient, contributing up to 66% reduction in terms of candidate named entities for disambiguation and up to 29% savings in CPU time compared against exhaustive Table Interpretation methods.

The remainder of this paper is organized as follows. Section 2 introduces key terms to be used in this article. Section 3 discusses related work. Sections 4 to 10 introduce TableMiner in details. Section 11 describes evaluation and discusses results, followed by conclusion in Section 12.

2. Notions and Definitions

Semantic Table Interpretation discovers implicit semantics in tabular data by addressing three tasks in the context of relational tables with reference to certain knowledge bases: Named Entity Disambiguation, Named Entity Classification, and Relation Enumeration.

A **relational table** T contains regular rows and columns resembling tables in traditional databases. Though in practice, tables can contain complex structures such as vertical relational tables and nested structures, they constitute a small population and are not the focus in this task. Nevertheless, work that handles parsing complex table structures can be found in, e.g., Zanibbi et al. [33]

Relational tables may or may not contain a **header row**, which is typically the first row in a table. They often contain a **subject column** that usually (but not necessarily) corresponds to the 'primary key' columns in a database table [30,31]. This contains the set of named entities the table is about (**subject entities**, e.g., column 'Title' in Figure 1 contains a list of films the table is about), while other columns contain either named entities forming binary relationships with subject entities, or literals describing properties of subject entities.

A **knowledge base** O defines a set of **concepts** $\mathcal{C}$; their object instances or **named entities** $\mathcal{E}$; **literals**

$\mathcal{L}$ representing concrete data values; and semantic **relations** $\mathcal{R}$ that define possible associations between named entities (hence also between concepts they belong to), or between a named entity and a literal, in which case the relation is usually called a **property** of the entity (hence a property of its concept) and the literal as the property value. In the generic form, a knowledge base is a liked data set containing a set of **statements, facts**, or **triples**, in the form of $<s, p, o>$, where the subject s could be a concept or named entity, the object o could be a concept, named entity, or literal, and the predicate p could be a relation or property. A knowledge base can be a populated ontology, such as the YAGO[1] and DBpedia[2] datasets, in which case a concept hierarchy is defined based on level of generalization. However this is not always true as some knowledge bases do not define strict ontology but loose concept networks, such as Freebase[3].

To recapitulate, given T and O, semantic Table Interpretation serves three goals (in no particular order): Named Entity Disambiguation associates content cells in NE-columns with one canonical named entity from $\mathcal{E}$; column classification labels each NE-column with one concept from $\mathcal{C}$, or in the case of literal-columns, associates the column to one property of the concept assigned to the subject column of T; Relation Extraction (or enumeration) identifies binary relations (from $\mathcal{R}$) between NE-columns, or in the case of one NE-column labeled by a concept c and another literal-column, the property of c that the latter represents (if any).

3. Related Work

A number of related fields of research are discussed in this section, including work on converting legacy structured data to linked data, NLP and IE in general, and semantic Table Interpretation which this work focuses on.

3.1. Legacy structured data to linked data

Research on converting tabular data in legacy data sources to linked data format has made solid contribution towards the rapid growth of the LOD cloud in the past decade [11,18,26,6]. Their difference from semantic Table Interpretation is that the focus is on data 'generation' rather than 'interpretation', since the goal is to pragmatically convert tabular data from databases, spreadsheets, and similar data structures into RDF. Typical methods require manually (or partially automated) mapping the two data structures (input and output RDF), and they do not link data to existing concepts, named entities and relations from the LOD cloud. As a result, the implicit semantics of the data remain concealed.

3.2. General NLP and IE

Due to the close ties with several classic research problems in the general NLP and IE domain, some may argue to use the extensively developed classic NLP/IE methods for Table Interpretation. In most cases, this is infeasible for a number of reasons. First, state-of-the-art methods [27,16] are typically tailored to unstructured text content that is different from tabular data. The interdependency among the table components cannot be easily taken into account in such methods [20]. Second and particularly for Named Entity Classification and Relation Extraction, each target semantic label (i.e., concept or relation) must be pre-defined and a learning process based on training or seed data is essential [25,34]. In Table Interpretation however, due to the large degree of variations in table schemata (e.g., Limaye et al. [19] use a dataset of over 6,000 randomly crawled Web tables of which no information about the table schemata is know a-priori), defining a comprehensive set of semantic concepts and relations and subsequently creating necessary training or seed data are infeasible.

Another marginally related IE task tailored to structured data is Wrapper induction [17,8], which automatically learns wrappers that can extract information from regular, re-occurring structures. It builds on the hypothesis that the same type of information are typically presented in similar structures in different documents and exploits such regularities to extract information. For this reason, it usually focuses on mining named entity attributes (which could be concepts or properties) from 'detail' webpages each describing a single named entity. In the context of relational tables, wrapper induction methods can be adapted to label table columns that describe named entity attributes. However, they also build on training data and the table schemata must be known a-priori. As discussed above, these are infeasible in the case of semantic Table Interpretation.

[1] http://www.mpi-inf.mpg.de/yago-naga/yago/
[2] http://wiki.dbpedia.org/Ontology
[3] http://www.freebase.com

3.3. Semantic Table Interpretation

Hignette et al. [13,14] and Buche et al. [1] propose methods to identify concepts represented by table columns and detect relations present in tables in a domain-specific context. An NE-column is labeled based on two factors: similarity between the header text of the column and the name of a candidate concept; plus the aggregation of the similarities calculated for each cell in the column and each term in the hierarchical paths containing the candidate concept. Authors also classify numeric data columns into domain-specific data types such as pH level and temperature, in a similar way. For relations, they only detect the presence of semantic relations in the table rather than specifying the columns forming binary relations. Scoring candidate relations also depends on two factors: the similarity between the name of a candidate relation and the table caption; plus the faction of rows that have been annotated by the two concepts connected by the relation.

Venetis et al. [30] annotate table columns and identify relations between the subject column and other columns using labels and relations from a database constructed by mining the Web using lexico-syntactic patterns such as the Hearst patterns [12]. The database contains co-occurrence statistics about the subject and object of triples. For example, how many times the word 'cat' and 'animal' has been extracted by the pattern *<?, such as ?>* representing the *is-a* relation between concept and instances. A maximum likelihood inference model predicts the best label for a column to be the one maximizing the probability of seeing all the values in the column given that label for the column. Such probability is computed based on the co-occurrence statistics gathered in the database. Relation interpretation follows the same principle.

Likewise, Wang et al. [31] argue that tables describe a single named entity type (concept) and its attributes and therefore, consist of a named entity column (subject column) and multiple attribute columns. The goal is to firstly identify the named entity column in the table, then based on the cell content on each row of the subject column and their corresponding values on the same row from other columns, associate a concept from the Probase knowledge base [32] that best describes the table schema. Essentially this allows labeling the subject NE-column and literal-columns using properties of the concept, also identifying relations between the subject column and other columns. Probase is a probabilistic database built in the similar way as that in Venetis et al. [30] 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.

Muñoz et al. [21] extract RDF triples for relational tables from Wikipedia articles. The content cells in the tables are required to have internal links to other Wikipedia articles. Such cells are firstly mapped to DBpedia named entities based on the links, then to derive relations between two named entities on the same row and from different columns, the authors query the DBpedia dataset for statements in the form of $<s, ?, o>$, where s and o are replaced by the two mapped named entities. Any predicates found in the query result are considered as relations between the two named entities.

Zwicklbauer et al. [39] use a simple majority vote model for column classification. Each candidate named entity in the content cells of the column casts a vote to the concept it belongs to, and the one that receives most votes becomes the concept label for the column. They show that for this very specific task, it is unnecessary to exhaustively disambiguate each content cell in a column. Instead, comparable accuracy can be obtained using a fraction of the content cells from the column. However, the sample size is empirically decided.

Syed et al. [29] deal with all three subtasks using DBpedia and Wikitology [28], the latter of which contains an index of Wikipedia articles describing named entities and a classification system that integrates several vocabularies including the DBpedia ontology, YAGO, WordNet[4] and Freebase. The method begins by firstly labeling each NE-column based on candidate named entities from every cell in the entire column. Candidate named entities for each content cell are retrieved by composing structured queries to match the cell text, the row content, and the column header against different fields defined in the Wikitology index, such as the title and redirects, the first sentence and links from candidate named entities' original Wikipedia articles. Candidate concept labels for the column combines the types associated with each candidate named entity from each cell, and the scores are based on the number of candidate named entities associated with that concept and the relevance score

[4] http://wordnet.princeton.edu/

of candidate named entities in the search results returned by Wikitology. The column labels are then used as input in named entity disambiguation, which is cast as queries to Wikitology with new constraints using the column label. Finally, relations between two NE-columns are derived based on the similar method by Muñoz et al. [21] using DBpedia. This method is later used in Mulwad et al. [24] and Mulwad et al. [22].

Limaye et al. [19] propose to model table components and their interdependence using a probabilistic graphical model. The model consists of two components: *variables* that model different table components, and *factors* that are further divided into node factors modeling the compatibility between the variable and each of its candidate, and edge factors modeling the compatibility between the variables believed to be correlated. For example, given an NE-column, the header of the column is a variable that takes values from a set of candidate concepts; and each content cell in the column is a variable that takes values from a set of candidate named entities. The node factor for the header could model the compatibility between the header text and the names of each candidate concept; while the edge factor could model the compatibility between any candidate concept for the header and any candidate named entity from each content cell. The strength of 'compatibility' could be measured using methods such as string similarity metrics [10] and semantic similarity measures [36]. Then 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 that it solves the three subtasks simultaneously, capturing interdependence between various table components at inference, while others either tackle individual subtasks or tackles all separately and sequentially.

Although the key motivation of using joint inference is to boost the overall quality of the labels, later work shows that they do not necessarily outperform models that deal with separate tasks independently [30]. Furthermore, Mulwad et al. [23] argue that computing the joint probability distribution in the model is very expensive. Thus built on the earlier work by Syed et al. [29] and Mulwad et al. [24,22], they propose a lightweight semantic message passing algorithm that applies inference to the same kind of graphical model. This is the most similar method to TableMiner, in the way that the *UPDATE* phase of TableMiner can be considered as a similar semantic message passing process. However, TableMiner is fundamentally different since it (1) uses bootstrapping learning designed to be efficient while Mulwad et al. build a model that approaches the task in an exhaustive way; (2) defines and uses context around tables as features while Mulwad et al. has used knowledge base specific features; (3) uses different scoring functions for candidate named entities, concepts and relations; and (4) models interdependence differently which, if transforms to an equivalent graphical model, would result in fewer factor nodes on the graph hence converges fast.

3.4. Remark

Existing Table Interpretation methods still suffer from several limitations to be detailed below.

Efficiency According to Limaye et al. [19], running the actual inference algorithm only accounts 1% of computation, while the large majority would be attributed to candidate search, data retrieval from knowledge bases and feature space construction. While except Zwicklbauer et al. [39], existing methods typically adopt an *exhaustive strategy* that examines the entire table content, e.g., column classification depends on every content cell in the column. As discussed before this is unnecessary in terms of human cognition as humans are able to make inference based on partial data. TableMiner uses bootstrapping based learning that begins by creating initial column classification using automatically determined data sample. The result from this process then guides disambiguation of the content cells by constraining candidate named entity space using the concept label assigned to the column. An iterative optimization process then enforces semantic interdependence across columns and between column classification and named entity disambiguation. The end outcome of such a design are reduction in candidate space and hence potential computational savings. Compared against Zwicklbauer et al. [39], TableMiner is able to automatically determine the sample size while Zwicklbauer et al. make arbitrary decisions; TableMiner tackles all three tasks of Table Interpretation while Zwicklbauer et al. only deal with column classification.

Features Table Interpretation methods have predominantly focused on features derived from knowledge bases and in-table context such as column header text and row content. However, as discussed before, out-table context such as table captions, webpage titles and paragraphs around tables offer useful clues for semantic interpretation. TableMiner proposes a generic model that is able to use any of such context as features in Table Interpretation. In addition, a large num-

ber of existing methods use knowledge base specific features that are non-generalizable. For example, the co-occurrence statistics used by Venetis et al. [30] and Wang et al. [31] are unavailable in knowledge bases such as YAGO, DBpedia, and Freebase. The search result relevance based features used in Syed et al. [29], Mulwad et al. [24,22,23] and Wang et al. [31] are specific to their carefully crafted knowledge bases. Methods such as Limaye et al. [19] use the concept hierarchy in YAGO; however Freebase does not have a strict concept hierarchy, but rather a simple concept network. TableMiner does not use any of such features and is therefore, highly generic.

Completeness As seen in the previous discussion, many Table Interpretation methods only deals with one or two subtasks. 'Full' interpretation methods include Limaye et al. [19], Syed et al. [29] and Mulwad et al. [24,22,23], which only handles NE-columns. TableMiner on the other hand, deals with all three subtasks, although relation interpretation is only limited to relations between the subject column and other columns in the table. This is because the majority of relational tables on the Web contains a subject column that forms binary relations with other columns [30]. These relations can be considered more important to the table than relations between non-subject columns. Moreover, TableMiner is capable of interpreting both NE- and literal-columns.

This work builds on our previous work [35][5] and largely extends it by: 1) significantly refining the column classification and named entity disambiguation methods by changing mathematical formulations and redefining the bootstrapping approach to better model interdependence between the two tasks; 2) adding subject column detection and relation enumeration; 3) carrying out substantially larger experiments to provide further insights to the method and releasing the largest Table Interpretation dataset.

4. Overview and Notations

Figure 2 shows the high-level view of TableMiner. Given a relational table it firstly detects a subject column (Section 6), which will be used for column interpretation and relation enumeration. Then TableMiner interprets NE-columns, coupling column classification with named entity disambiguation in an incremental, mutually recursive, bootstrapping approach that consists of a *LEARNING* phase (Section 8) and an *UPDATE* phase (Section 9). The *LEARNING* phase interprets each column independently. It begins by learning initial column classification using partial content from the column (*sample based classification*). The results are then used to constrain named entity candidate space in disambiguating content cells in the column (*constrained disambiguation*). The *UPDATE* phase enforces interdependence between columns, and between the classification and disambiguation results. This is achieved by an iterative optimization process where in each turn, column labels are revised against a notation of 'domain consensus' that is derived from disambiguated named entities from all columns; then named entity annotations are revised due to (if any) the change of column labels. The process repeats until results are stabilized (i.e., convergence), creating a global optimum. Finally relation enumeration (Section 10) discovers binary relations between the subject column and other NE-columns; or identifies a property of the subject column's concept to best describe data in the literal-columns. In the latter case, the property is considered both the label for the concerning literal-column, and its relation with the subject column.

Both subject column detection and initial column classification in the *LEARNING* phase are based on automatically determined sample data from the table (columns). This is done by using an *incremental inference with stopping* algorithm (*I-Inf*), which is discussed in Section 5.

At different steps, TableMiner uses features from both in-table and out-table context listed in Table 1. In particular, out-table context includes **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, relations, or descriptions of entities in the table. Furthermore, an increasing amount of linked data is becoming available in the format of within-webpage semantic annotations using markup vocabularies (e.g., *schema.org*) heavily promoted by major search engines [4]. 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

[5] Currently under review.

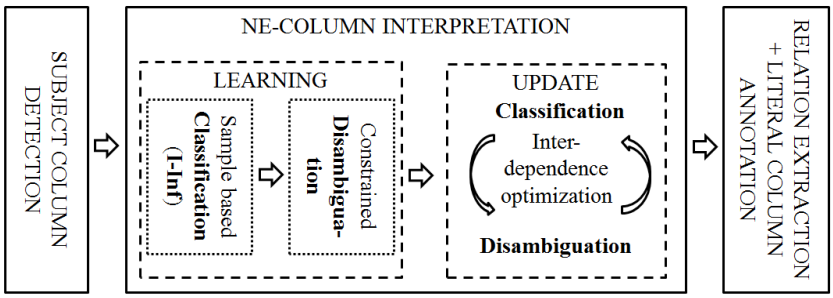

Fig. 2. The overview of TableMiner. Arrows indicates that the output from one process is passed as input to the next.

Rich Snippet[6] to improve content access. Such data provides important clues on the 'aboutness' of a webpage, and therefore tables (if any) within the webpage. Details of how such features are used are dependent on subtasks and will be discussed in the following sections.

Table 1
Types of context from which features are created for Table Interpretation.

In-table context	Out-table context
column header	webpage title
row content	table caption and/or title
column content	paragraphs (unstructured text)
	semantic markups

4.1. Notations

The following notations will be used in the discussion. Let T denotes a relational table containing i rows and j columns of *content cells*, i.e., i does not index the header row. T_i denotes content row i, T_j denotes column j, TH_j denotes the header of column j, and $T_{i,j}$ denotes cell (i, j). X denotes a set of *context* where features can be derived (Table 1).

$\mathcal{C}^T$ contains elements C_j the candidate labels with scores for column T_j in Table T. A candidate label for an NE-column is a concept and for a literal-column a property of the concept assigned for the subject column in the table (e.g., in Figure 1 'Year' matches the property 'releaseDate' of the concept 'Film' assigned to the first column). $\mathcal{E}^T$ contains elements $E_{i,j}$, which maps to each content cell in T and contains a set of candidate named entities (if any) for the cell $T_{i,j}$. $\mathcal{R}^T$ contains elements $R_{j,j'}$, which is a set of candidate relations between the subject column j and column j' ($j \neq j'$). All candidate concepts, named entities, and relations are retrieved from a knowledge base O.

Function $l(o)$ returns the 'name' or 'label' of object o. o can be a table component object, in which case the string content inside the component is returned (e.g., in Figure 1 $l(TH_1)$ ='Title'). It can also be an annotation, i.e., $c^n_j, e^n_{i,j}$, or $r^n_{j,j'}$, in which case the name and the id of the annotation are concatenated. For example, given the named entity identified by an URI id '`http://dbpedia.org/John_Smith_(Conservative_politician)`' with a label 'Smith, John', the concatenated string is 'Smith, John, John Smith Conservative politician'[7].

Unless otherwise defined, function $bw(o)$ returns a bag-of-words (multiset) of an object by tokenizing $l(o)$, normalizing tokens by morphological analysis then removing stopwords and words containing a single character. $bwset(o)$ returns the set of unique tokens in $bw(o)$. $|\cdot|$ returns the number of elements in a collection (either containing duplicates or de-duplicated). w denotes a single normalized token. $freq(w, o)$ counts the frequency of w in $bw(o)$.

5. I-Inf

The ***I**ncremental **Inf**erence with stopping* algorithm is used by both subject column detection and column classification. Shown in Algorithm 1, it is an iterative process where in each turn, a single data item is processed and updates the overall state until convergence. The $state$ is a map containing tuples es =<*entry, score*>, where the $entry$ set contains candidate objects we wish to evaluate by $score$. At each iteration, the $performTask$ function processes a data item

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

[7] URIs are trimmed by removing the domain.

$d \in D$ and generates new tuples based on d. This updates $state$ by resetting scores of existing $entries$ or adding new $entries$. The specific definitions of $state$, $performTask$ and d will be given in the following sections.

At the end of each iteration, *I-Inf* compares the current $state$ with the previous to check for convergence. To do so, the $entropy$ is computed for each $state$ as below, and convergence happens if the difference between the current and previous states is less than a threshold t.

Algorithm 1 I-Inf

```
1: Input: D, state ← ∅
2: Output: the collection of es from state ranked by score
3: for all d ∈ D do
4:    prev ← state
5:    {es_i} ← performTask(d)
6:    update(state, {es_i})
7:    if convergence(state, prev) then
8:       break
9:    end if
10: end for
```

$$entropy(state) = -\sum_{es_i \in state} P(es_i) \log_2 P(es_i) \tag{1}$$

$$P(es) = \frac{es_{score}}{\sum_{es' \in state} es'_{score}} \tag{2}$$

, where es_{score} returns $score$ in the tuple *<entry, score>*. The intuition is that when the entropy level converges, we expect $P(es_i)$ for each i to also converge. This could suggest that the processing of additional data items changes little the $score$ of each es_i with respect to the sum in the state (i.e., the denominator in Equation 2). Effectively this means that although the absolute score of each entry still changes upon additional data items, their relative scores may not and hence their rankings tend to stabilize.

6. Subject column detection

Venetis et al. [30] introduce a supervised subject column classifier using several features derived from tables. This work introduces an unsupervised subject column detection algorithm that uses a different set of features listed in Table 2.

6.1. Preprocessing

Subject column detection begins by classifying each column's content cells into one of the data types: *empty*, *named entity*, *number*, *date expression*, *long text* (e.g., sentence, or paragraph), and *other*. This is done by using simple regular expressions that examine the syntactic features of cell text, such as number of words, capitalization, mentions of months or days in a week. The regular expressions focus on maximizing recall. A vast amount of literature can be found on this topic. An example of recognizing cells containing named entities can be found in Hignette et al. [13,14] and Buche et al. [1]. Then each column is assigned a most frequent datatype by counting the number of content cells belonging to that type. The only exception is that a column is 'empty' only if all content cells are empty.

Next, if a candidate NE-column has a column header that is a preposition word, it is discarded. An example like this is shown in Figure 3, where the columns 'For' and 'Against' are clearly not subject columns but rather form relations with the subject column 'Batsman'.

6.2. Features

Next, features listed in Table 2 are constructed for each remaining candidate NE-column. The fraction of **empty cells** (ec) of a column is simply the number of empty content cells divided by the number of rows in the column. Likewise, the fraction of cells with **unique content** (uc) is a ratio between the non-duplicate cell content in the column and the number of rows. The **distance from the first NE-column** (df) counts how many columns the current column is away from the first candidate NE-column from the left. NE-columns are also checked by regular expressions to identify the number of content cells likely to contain acronyms or ids such as airline IACO codes (e.g., using features like uppercase letters and presence of white spaces). A column that is an **'acronym' or 'id' column** ($ac(T_j) = 1$, or 0 otherwise) is disfavored. The intuition is that as the subject of a table one would prefer to use full names of entities for the sake of clarity.

Context match score (cm) for a column T_j is based on the frequency of the column header found in the table's context:

$$cm(T_j) = \sum_{x_j \in X_j} \sum_{w \in bw(TH_j)} freq(w, x_j) \times wt(x_j) \tag{3}$$

Table 2
Features used for subject column detection. Examples are based on the visible content in Figure 1

Feature	Notation	Example
Fraction of empty cells	ec	0.0 for column 'Title'
Fraction of cells with unique content	uc	1.0 for column 'Title', 5/8 for column 'Director'
If >50% cells contain acronym or id	ac	-
Distance from the first NE-column	df	0 for column 'Title', 2 for column 'Director'
Context matching score	cm	-
Web search matching score	ws	-

Rank	Batsman	Score	For	Against	Venue	Season	Rating
1	Donald Bradman	270	Australia	England	Melbourne	1936–37	262.4
2	Brian Lara	153*	West Indies	Australia	Bridgetown	1998–99	255.2

Fig. 3. An example table containing columns with preposition words as headers.

, where $x_j \in X_j$ are context of the column header TH_j, and $wt(x_j)$ is the weight given to a specific type of context. Intuitively, the more frequent a column header text is repeated in the table's context the more likely it is the subject column of the table. The context elements used for cm include *webpage title*, *table caption* and *surrounding paragraphs*.

Web search score (ws) exploits information from the Web to infer a subject column. Given each row T_i, a query string q is firstly created by concatenating all $l(T_{i,j})$, where j must correspond to NE-columns. For example, q ='Love and Larceny, Dino Risi, Vittorio Gassman' is created for row #4 in the table from Figure 1. Then q is queried on a search engine to retrieve the top n[8] webpages. Let PG denotes these webpages, then each NE-cell $T_{i,j}$ that composes q is scored as:

$$ws(T_{i,j}) = countp(l(T_{i,j}), PG) + countw(l(T_{i,j}), PG) \quad (4)$$

$$countp(l(T_{i,j}), PG) = \sum_{pg \in PG} (freq(T_{i,j}, pg_{ttl}) \times 2 + freq(T_{i,j}, pg_{snp})) \quad (5)$$

$$countw(l(T_{i,j}), PG) = \sum_{pg \in PG} \sum_{w \in bwset(T_{i,j})} \frac{freq^{-}(w, pg_{ttl}) \times 2 + freq^{-}(w, pg_{snp})}{|bw(T_{i,j})|} \quad (6)$$

$countp$ sums up the frequency of the entire cell text in both the titles (pg_{ttl}) and snippets (pg_{snp}) of returned webpages. Frequency in titles are given double weight as they are considered to be more important. $countw$ partitions cell text into tokens ($bw(T_{i,j})$), then counts frequency of each unique token. $freq^{-}$ ensures those occurrences of w that are included within occurrences of $l(T_{i,j})$ are eliminated. For example, we do not double-count 'Dino' and 'Risi' where the two words occur as an entire phrase hence counting as one occurrence of $l(T_{i,j})$. Instead only separate occurrences of 'Dino' or 'Risi' are counted. The frequency is then normalized by the size of $bw(T_{i,j})$.

For each row, the cell that receives the highest score is considered to be containing the subject named entity for the row. The intuition is that the query q should contain the name of the subject entity plus contextual information of what the named entity is 'about'. When searched, it is likely to retrieve more documents regarding the subject named entity than its 'aboutness', and the subject named entity is also expected to be repeated more frequently. Finally the ws score of a column $ws(T_j)$ is simply the summation of $ws(T_{i,j})$ for every row i in the column.

I-Inf for Web search score In principle, Web search score should be computed for each row of a table to de-

[8]The default of a search engine's API is used.

rive the final column scores $ws(T_j)$. Such an exhaustive strategy is practically inefficient, and in fact unnecessary. Again using Figure 1 as an example, one does not need to read all 60 rows to decide the column 'Title' as the best subject column candidate among others. Therefore, computing Web search scores is wrapped by the *I-Inf* algorithm. Specifically in Algorithm 1, D is replaced by the collection of table rows T_i, each tuple in $state$ is defined as $<T_j, ws(T_j)>$, i.e., a column and its ws score. In each iteration, one additional row T_i is processed ($performTask$) and the $state$ is updated with corresponding results from the row. Specifically, the ws score of a column T_j is changed due to the summation of $ws(T_{i,j})$ for every row that has been processed by the current iteration. This process repeats until convergence, when the column scores are finalized.

6.3. Detection

Features (except *df*) are then normalized into relative scores by the maximum score of the same feature type.

$$ft_{norm}(T_j) = \frac{ft(T_j)}{\max_{\forall \textit{NE-column } j'} ft(T_{j'})} \quad (7)$$

, where ft denotes a feature (e.g., cm, ws). Next, they are combined to compute a final subject column score $subcol(T_j)$ and the column with the highest score is chosen as subject column.

$$subcol(T_j) = \frac{uc_{norm}(T_j) + 2(cm_{norm}(T_j) + ws_{norm}(T_j)) - ec_{norm}(T_j)}{\sqrt{df(T_j) + 1}} \quad (8)$$

uc, cm and ws are all indicative features of subject column in a table. However, uc is given half the weight of cm and ws, because subject columns do not necessarily contain unique values at every row [30], but they are more likely than not. While cm and ws are intuitively stronger indicators. A column that contains empty cells is penalized, and the total score is normalized by its distance from the left-most NE-column as subject columns tend to appear on the left and before 'attribute' columns.

7. NE-Column interpretation - building blocks

Then given an NE-column, the interpretation consists of a *LEARNING* phase that creates initial classification and disambiguation in each column independently from others, and an *UPDATE* phase that iteratively optimizes the results by enforcing interdependence. Both processes require two basic operations: (1) computing candidate named entity score and (2) computing candidate concept score.

7.1. Computing candidate named entity score

Given a candidate named entity $e^n_{i,j} \in E_{i,j}$ for a cell $T_{i,j}$, the disambiguation score $disamb(e^n_{i,j})$ depends on two components: **context** scores between $e^n_{i,j}$ and each type of its context $x_{i,j} \in X_{i,j}$, and a **name match** score between $l(e^n_{i,j})$ and $l(T_{i,j})$.

The **similarity** between $e^n_{i,j}$ and any $x_{i,j} \in X_{i,j}$ is computed based on the overlap between the $bw(e^n_{i,j})$ and $bw(x_{i,j})$. In this case to build $bw(e^n_{i,j})$, the *triples* {<*s,p,o*>}about $e^n_{i,j}$ is retrieved from the knowledge base. $bw(e^n_{i,j})$ is simply based on the concatenation of the object *o* from all triples.

$X_{i,j}$ is composed of all of the out-table and in-table context shown in Table 1. *Row content* concatenates $l(T_{i,j'})$ where $j \neq j'$. These are likely to be attribute data values of named entities. *Column content* concatenates $l(T_{i',j})$ where $i \neq i'$. These are likely to refer to named entities that are semantically similar or related. *Column header* could contain useful features as named entities sometimes use words that indicate its semantic type in its names (e.g., 'River Sheaf'). *Webpage title*, *table captions/titles* and surrounding *paragraphs* can contain words that are important to named entities. *Semantic markups* may annotate important named entities or their attributes on the webpage. They are represented as RDF triples and the objects of triples that are literals are concatenated. The overlap between $e^n_{i,j}$ and any type of out-table context is computed using a frequency weighted dice function, while for in-table a function measuring 'coverage' because intuitively the presence of any in-table features in the bw representation of a candidate named entity is a stronger signal of what the named entity is about.

$$dice(e^n_{i,j}, x_{i,j}) = \frac{2 \times \sum_{w \in bwset(e^n_{i,j}) \cap bwset(x_{i,j})} (freq(w, e^n_{i,j}) + freq(w, x_{i,j}))}{|bw(e^n_{i,j})| + |bw(x_{i,j})|} \quad (9)$$

$$coverage(e^n_{i,j}, x_{i,j}) = \frac{\sum_{w \in bwset(e^n_{i,j}) \cap bwset(x_{i,j})} freq(w, x_{i,j})}{|bw(x_{i,j})|} \quad (10)$$

Then let $sim(e^n_{i,j}, x)$ be the generalized function (either $dice$ or $coverage$) that measures similarity between a named entity and its context $x_{i,j}$, the weighted sum of the similarity between $e^n_{i,j}$ and each $x_{i,j} \in X_{i,j}$ becomes the overall context similarity score $ectx(e_{i,j}, X_{i,j})$.

$$ectx(e_{i,j}, X_{i,j}) = \sum_{x_{i,j} \in X_{i,j}} sim(e_{i,j}, x_{i,j}) \times wt(x_{i,j}) \quad (11)$$

The **name match** score between $e^n_{i,j}$ and $T_{i,j}$ is computed based on the standard Dice coefficient as:

$$nm(e^n_{i,j}, T_{i,j}) = \sqrt{\frac{2 \times |bwset(l(e^n_{i,j})) \cap bwset(T_{i,j})|}{|bwset(l(e^n_{i,j}))| + |bwset(T_{i,j})|}} \quad (12)$$

Finally, an overall disambiguation score $fse(e^n_{i,j})$ is computed using Equation 13 below. The factor $\sqrt{|bw(T_{i,j})|}$ balances the weight between nm and $ectx$ by the number of tokens in $bw(T_{i,j})$. Intuitively, a named entity mention that is a long name consisting of multiple tokens (e.g., 'Harry Potter and the Philosopher's Stone') is less likely to be ambiguous than a single-token name ('Harry'). Therefore the nm score in the former case should be given higher weight (or conversely, $ectx$ is given less weight). When the mention has only a single token, both nm and $ectx$ are given the equal weight.

$$fse(e^n_{i,j}) = nm(e^n_{i,j}, T_{i,j}) + \frac{ectx(e^n_{i,j}, X_{T_{i,j}})}{\sqrt{|bw(T_{i,j})|}} \quad (13)$$

7.2. Computing candidate concept score

The set of candidate concepts for a column C_j is derived based on the highest scoring (i.e., winning) candidate named entities in its containing cells:

$$C_j \leftarrow \bigcup_{i \in I} cls(\arg\max_{e^n_{i,j}} fse(e^n_{i,j})) \quad (14)$$

, where function $cls(e^n_{i,j})$ returns the set of concepts that the named entity belongs to, I is the set of row indexes to be considered. In case of a sample is used, I includes a subset of rows in T, otherwise I simply denotes all rows in T. Then the score of each candidate concept $c^n_j \in C_j$ consists of two parts: (1) a **base score** depending on its contributing named entities, and (2) a **context score** based on its context.

To compute the **base score** bs, let $disamb(c_j)$ be the sum of $fse(e^n_{i,j})$ where $e^n_{i,j}$ is the winning named entity from any row that selects c^n_j, then:

$$bs(c^n_j) = \frac{disamb(c^n_j)}{|I|} \quad (15)$$

For a candidate c^n_j, when the winning named entity $e^n_{i,j}$ for every $i \in I$ selects c^n_j and each contributing $fse(e^n_{i,j})$ approaches to 1.0, its base score will approach to the highest of 1.0.

The concept's **context score** $cctx(c_j)$ is computed in the same principle as $ectx$. A similarity score between c^n_j and each of its context $x_j \in X_j$ is computed using the weighted dice function replacing $e_{i,j}$ with c_j and $x_{i,j}$ with x_j (as introduced in Section 4.1, $bw(c^n_j)$ is based on $l(c^n_j)$). The out-table context for a column header (hence the candidate concepts) includes all out-table context shown in Table 1. The in-table context excludes row content; and column content concatenates all content cells $T_{i,j}$, because entity names can contain indicative words of their semantic types.

The final score of a candidate concept $fsc(c^n_j)$ equally combines $bs(c^n_j)$ and $cctx(c^n_j)$, and the highest scoring candidate (i.e., winning concept) is the one chosen to label the column.

$$fsc(c^n_j) = bs(c^n_j) + cctx(c^n_j) \quad (16)$$

8. NE-Column interpretation - the LEARNING phase

The *LEARNING* phase of NE-column interpretation creates initial classification for the column and disambiguates named entity mentions in each content cell.

8.1. Sample-based classification with I-Inf

The process of initial classification is based on a preliminary disambiguation process applied only to the automatically selected sample from a column. The disambiguation result from this process is subject to revision in the next step and the ultimate goal is creating initial labels for the column. As the principle is to use *partial* content instead of the entire column for initial classification, the choice of sample items and the sample size may affect the learning accuracy. The first issue is addressed with a one-sense-per-discourse hypothesis and sample item ranking; the second issue is addressed by wrapping classification under the *I-Inf* algorithm.

8.1.1. Preliminary disambiguation

Given $T_{i,j}$, the text content $l(T_{i,j})$ is searched in a knowledge base to retrieve candidate named entities $E_{i,j}$. Candidates whose $l(e^n_{i,j})$ has no overlapping tokens (stop words ignored) with $l(T_{i,j})$ are discarded. Then each $e^n_{i,j} \in E_{i,j}$ is scored following the method described in Section 7.1, and the winning named entity is chosen for the cell $T_{i,j}$.

8.1.2. Sample item ranking and one-sense-per-discourse

One-sense-per-discourse is a common hypothesis in the literature of sense disambiguation. The idea is that that a polysemous word appearing multiple times in a well-written discourse is 'extremely likely' to share the same sense [7]. Though this is widely followed in sense disambiguation in unstructured texts, this work notes that this is also common in relational tables: given a *non-subject column*, multiple content cells containing the same text content are extremely likely to express the same meaning (e.g., same named entity or concept). Although in most cases this also applies to subject columns, it has been noted that they are more likely to violate such a rule than non-subject columns. As discussed before, subject columns do not always contain unique content at each cell, in which case they do however, express different meanings. A typical example is the Wikipedia article 'List of peaks named Bear Mountain'[9], which contains a disambiguation table listing different peaks sharing the same name 'Bear Mountain' (subject column).

In this work, given an NE-column T_j, the sample item ranking method simply promotes those content cells $T_{i,j}$ whose row content is 'rich'. This is assessed by a 'sample score' (ss) that counts the number of non-empty cells on the containing row T_i, where for any two cells $l(T_{i,j}) = l(T_{i',j})$, their row content are merged:

$$ss(T_{i,j}) = \sum_{\forall T_{i',j}: l(T_{i,j})=l(T_{i',j})} \sum_{j'} \begin{cases} 1 & \text{if } nonempty(T_{i,j'}) \\ 0 & otherwise \end{cases} \tag{17}$$

, where $nonempty(T_{i,j'})$ is true if the cell is not empty. The intuition is that the more contextual features we can extract for a cell, the more knowledge we acquire about it and therefore, the more confident we are about disambiguating candidate named entities of the cell. Therefore sample ranking promotes content cells with high sample scores, effectively favoring those grouped under the one-sense-per-discourse hypothesis.

This sample item ranking method is rather simplistic. Intuitively we would prefer to select items that best 'represent' the most suitable concept for the column, in practice it can be difficult to define 'representativeness' without a-priori knowledge about the concept. This will be further explored in future work.

8.1.3. Putting-it-together with I-Inf

In the context of *I-Inf*, the input D is an ordered list L of $T_{i,j}$ for a certain column T_j based on $ss(T_{i,j})$. The $state$ consists of candidate concepts and their final scores as tuples $< c^n_j, fsc(c^n_j) >$. Then in each iteration, the top-ranking $T_{i,j}$ is drawn and processed by $performTask$, which disambiguates the cell content following the method in Section 8.1.1. Then $update$ operation derives candidate concepts based on the winning named entity from already processed rows in the current iteration (Equation 14, note that I changes in each iteration, corresponding to content cells that are already drawn from L), and updates the $state$.

To do so, if a derived candidate concept does not exist in the current $state$, it is simply added; otherwise, the base score bs of the existing candidate concept is

[9] http://en.wikipedia.org/wiki/List_of_peaks_named_Bear_Mountain, last retrieved 13 April 2014.

revised to account for the newly disambiguated content cell, and the final score fsc is updated accordingly. This process repeats, incrementally processing the column and updating column classification until convergence (Section 5). Thus initial classification can be created before processing the entire column (particularly true in large tables, as shown by experiments), reducing computational overheads. The end output of this process is C_j^{init}, containing the winning candidate concepts (multiple candidates are possible) from C_j.

8.2. Constrained named entity disambiguation

Next, C_j^{init} are used as constraints to assist disambiguation in the column. The disambiguation of each content cell follows the same procedure as *preliminary disambiguation* in Section 8.1.1 with one exception: *candidate retrieval*. Using C_j^{init} only entities that belong to at least one concept in C_j^{init} are considered as disambiguation candidates. Note that for content cells already processed by *preliminary disambiguation*, there is no need for re-computation: the best candidate named entity for the cell is simply reselected as the winning candidate satisfying $cls(e_{i,j}^n) \cap C_j^{init} \neq \emptyset$. Compared to the typical exhaustive strategy adopted by state-of-the-art, *constrained disambiguation* reduces computation by cutting down the number of named entity candidates for consideration.

Each additional (to those already processed *preliminary disambiguation*) disambiguated content cell generates a set of concepts, of which some can be new while others may already exists in C_j. By the end of the disambiguation on the entire column, C_j is updated by: (1) adding newly derived concepts; (2) for those already exist in C_j at the beginning, revising their base scores - hence also the final score fsc - to account for new evidence. This changes C_j and in some cases, causes the resulting winning candidate concepts to be inconsistent from C_j^{init} at the beginning of disambiguation. This is handled by the *UPDATE* phase to be discussed in the following section.

9. NE-Column interpretation - the UPDATE phase

The *UPDATE* phase of NE-column interpretation enforces interdependence between classification and disambiguation *within* each NE-column as well as *across* different columns. This is done by an iterative optimization process shown in Algorithm 2.

Algorithm 2 UPDATE

```
Input: C^T, E^T, prev_C^T ← ∅, prev_E^T ← ∅
while stabilized(C^T, E^T, prev_C^T, prev_E^T)=false do
    prev_C^T ← C^T, prev_E^T ← E^T
    bw(domain) ← domainrep(E^T)
    for all C_j ∈ C^T do
        for all c_j^n ∈ C_j do
            fsc(c_j^n) = bs(c_j^n) + cctx(c_j^n) + dc(c_j^n, bw(domain))
        end for
        C_j^best ← selectBest(C_j)
        C_j, E_{i,j} ← disambiguate(T_{i,j}, C_j^best)
    end for
end while
```

Starting with $\mathcal{E}^T = \{E_{i,j} | \forall i,j\}$ the set of disambiguation candidates for all NE-content cells in the table and the set of candidate concepts $\mathcal{C}^T = \{C_j | \forall j\}$ for all NE-columns, the update process computes a *domain consensus* dc score for each candidate concept and updates their final score fsc by adding dc. A domain representation $bw(domain)$ is created based on the winning candidate named entity in every NE-content cell:

$$bw(domain) \leftarrow \bigcup_{i,j} shortbw(\arg\max_{e_{i,j}^n} fse(e_{i,j}^n)) \tag{18}$$

, where $shortbw$ takes a definitional sentence about a named entity and converts it into bag-of-words representation in the same way as $bw(o)$. A definitional sentence is commonly found in almost any knowledge base. For example, WordNet has a one-sentence definition for every synset; the first sentence in an Wikipedia article usually defines an entity [15]; this also applies to the description of a Freebase topic (e.g., a concept or named entity) or a DBpedia resource. The idea is that the definitional sentence provides a focused description of the named entity, likely to contain informative words about the general domain it is related to. In particular, it often contains words forming hypernymy relation with the named entity [15,37]. For example, the Freebase definitional sentence about the English city 'Sheffield' contains words[10] 'city', 'metropolitan', 'borough', 'Yorkshire', and 'England', which are useful words defining the concept space of the named

[10] http://www.freebase.com/m/0m75g, last retrieved on 13 April 2014.

entity. Then dc is computed based on the frequency weighted dice function (Equation 9):

$$dc(c_j, domain) = \sqrt{dice(c_j, domain)} \qquad (19)$$

Since the sizes of $bw(c_j)$ and $bw(domain)$ are often different orders of magnitude, square root is used to re-balance the score. dc is computed for each candidate concept on each column, then added to the final score (Line 7 in Algorithm 2). Also dc is computed with respect to named entities from all content cells in any columns, which serves as a way of ensuring inter-column dependence.

After revising the final scores of candidate concepts, the set of winning candidates for each column are re-selected (Line 9). If for any column, this is different from the that generated in the previous iteration, the disambiguation result on that column is revised (Line 10). This follows the procedure of *constrained named entity disambiguation* (Section 8.2) using C_j^{best} as constraints.

These updating processes could change $\mathcal{C}^T$ and $\mathcal{E}^T$. Function $stabilized$ then derives the winning candidate concepts for every NE-column from $prev\_\mathcal{C}^T$ and compare them against those derived from $\mathcal{C}^T$; further, it also derives the winning candidate named entity at each content cell from $prev\_\mathcal{E}^T$ and compares them against those derived from $\mathcal{E}^T$. The update process is called to be 'stabilized' if both comparisons detect no difference. Otherwise, a new iteration repeats until stabilization.

Note that re-computing disambiguation and classification may require retrieving data of new named entity candidates from the knowledge base and subsequently constructing their feature representation for disambiguation due to the change of C_j^{best} at each iteration. However, this design still largely improves the exhaustive strategy because (1) empirically the *UPDATE* phase converges fast and in most cases involves merely re-selecting those 'losing' candidate named entities that were already seen in the *LEARNING* phase; (2) when new candidates are indeed added it only happens in significantly fewer content cells than the entire column that an exhaustive strategy would otherwise have to deal with.

10. Relation enumeration and labeling literal-columns

10.1. Relation enumeration

Relation enumeration firstly begins by interpreting relations between the subject column (T_j) and any other columns (T'_j, either NE- or literal-columns) on *each row* independently.

Let $e_{i,j}^{best}$ be the winning subject named entity for $T_{i,j}$ from disambiguation, and $T_{i,j'} (j \neq j')$ is another content cell on the same row. The candidate set of relations between $T_{i,j}$ and $T_{i,j'}$, denoted by $R_{j,j'}^i$, is derived from the triples containing $e_{i,j}^{best}$ as subject, denoted by $TP_{i,j} = \{< e_{i,j}^{best}, p, o >\}$. Then let $tp_{i,j} \in TP_{i,j}$ be an instance of the triples, and functions $s(tp_{i,j}), p(tp_{i,j}), o(tp_{i,j})$ return the subject, predicate and object from the triple respectively, the candidate relations $R_{j,j'}^i$ is the set of unique predicates in $TP_{i,j}$, i.e., $\{p(tp_{i,j}) | \forall tp_{i,j} \in TP_{i,j}\}$.

Then a confidence score rs is computed for each candidate relation $r_{j,j'}^i \in R_{j,j'}^i$ and the highest scoring (winning) candidate relation is chosen for the row. To compute the confidence score, the subset of triples from $TP_{i,j}$ containing $r_{j,j'}^i$ as predicate are selected. Then the object of each triple in the subset is matched against the content in $T_{i,j'}$ by string similarity, and the highest score is assigned to be rs:

$$rs(r_{j,j'}^i) = \max_{tp_{i,j} \in TP_{i,j}, p(tp_{i,j}) = r_{j,j'}^i} simlex(T_{i,j'}, o(tp_{i,j})) \qquad (20)$$

, where function $simlex(T_{i,j'}, o(tp_{i,j}))$ computes a string similarity score between the bag-of-words representations (i.e., using function bw) of $T_{i,j'}$ and $o(tp_{i,j})$ using the standard Dice coefficient function. The winning candidate relation for the row is denoted by $wr_{j,j'}^i$ and is $\arg\max_{r_{j,j'}^i} rs(r_{j,j'}^i) \forall r_{j,j'}^i \in R_{j,j'}^i$.

After the winning candidate relation is computed for each row between T_j and T'_j, the candidate set of relations for the two columns $R(j, j')$ is derived by concatenating the winning candidates on all rows:

$$R(j, j') \leftarrow \bigcup_i wr_{i,j'}^i \qquad (21)$$

Then a confidence score is computed for each instance $r_{j,j'}^n \in R(j, j')$, and it consists of two parts: a **base score** rbs and a **context score** $rctx$, each computed in the similar way as for concepts. Let

$disamb(r^n_{j,j'})$ be the sum of $rs(wr^i_{j,j'})\forall i$ such that the winning relation between columns j and j' on row i is $r^n_{j,j'}$, and $rows(T)$ returns the number of rows in the table. The base score is computed as:

$$rbs(r^n_{j,j'}) = \frac{disamb(r^n_{j,j'})}{rows(T)} \quad (22)$$

The **context score** $rctx(r_{i,j})$ is computed in the same way as the context score for candidate concepts $cctx(c^n_j)$. $bw(r^n_{j,j'})$ is based on $l(r^n_{j,j'})$. The context of candidate relations includes *column header* (which sometimes indicate the relation with the subject column), surrounding *paragraphs* and *semantic markups*. Other types of context are less likely to contain mentions of relations.

The final score of a candidate relation $fsr(r^n_{j,j'})$ adds up its base score and context score with equal weights, and the final binary relation that associates subject column T_j with column T'_j is the candidate with the highest fsr score.

10.2. Labeling literal-columns

Literal-columns are expected to contain attribute data of named entities in the subject column. They do not denote named entities and therefore, cannot be interpreted using the column interpretation method described above. In previous work [19,29,24,22,23] they are simply ignored. This work also assigns a column label that best describes the attribute data in literal-columns.

Given a literal-column T'_j that forms a binary relation $r^n_{j,j'}$ with the subject column T_j, the column label for this column is simply $l(r^n_{j,j'})$, since $r^n_{j,j'}$ typically describes a property of the subject column concept in such cases.

11. Evaluation

This section introduces the datasets and methods for evaluating TableMiner, followed by results and discussions. Methods of Table Interpretation have been evaluated by both *in-vitro* (assessing the annotations directly) and *in-vivo* (assessing the accuracy of applications built on top of the annotations) experiments. This work evaluates TableMiner using *in-vitro* experiments because (1) they are the most commonly used evaluation approach and (2) standard datasets are available.

11.1. Knowledge base and datasets

In this work, Freebase is used as the knowledge base for Table Interpretation. Freebase is currently the largest well-maintained knowledge base and linked data set 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.

Four datasets[11] have been created to evaluate TableMiner: **Limaye200**, **LimayeAll**, **IMDB** and **MusicBrainz**.

11.1.1. LimayeAll and Limaye200

These datasets are the rebuilt versions of the original four datasets used by Limaye et al. [19]. The original datasets consist of over 6,000 tables, 94% collected from Wikipedia and the rest from the general Web. Named entities in the tables were annotated with links to Wikipedia articles; columns and binary relations between columns were annotated by concepts and relations in the YAGO knowledge base (2008 version).

These datasets are re-created for a number of reasons. First, Wikipedia has undergone significant changes since the publication of the datasets such that a large proportion of the source webpages - as well as their contained tables - have been changed. Second, it has been noted in this work the original ground truth for named entity disambiguation were very sparse and possibly biased. As shown in Appendix A, it is less well-balanced than the re-created dataset and a simple exact name match baseline has achieved significantly higher accuracy than the original reported results in Limaye et al. [19]. Third, this work uses a different knowledge base from YAGO, such that the original ground truths cannot be directly used.

Re-generating named entity ground truth - LimayeAll The original datasets are firstly divided into tables extracted from Wikipedia (wiki-table) and those from the general Web (Web-table). Each wiki-table is re-created based on the current version Wikipedia. To do so, the corresponding Wikipedia article is downloaded, and tables containing links to other Wikipedia articles (internal links) are extracted. Each newly extracted table is then submitted to a content checking process against the original table. Let T_n denote one such table from the new Wikipedia article and T_o denote the original table. $bw(T_n)$ and $bw(T_o)$ creates a bag-of-words representation of T_n and T_o respectively

[11] Download from: http://staffwww.dcs.shef.ac.uk/people/Z.Zhang/resources/tableminer/data.tar.gz

by concatenating all content cells and headers in each table then removing stop words. The similarity between the two tables is computed using the frequency weighted dice function in Equation 9 to obtain a score between 0 and 2. Then if the one T_n that has the highest similarity score satisfies the following conditions it is selected: (1) has a similarity score of greater than 1.0; (2) contains at least an equal number of rows as T_o and at least two columns; (3) has less than 200 rows[12].

If no tables are extracted from the new source article or pass the content check, the original table T_o is re-annotated by a 'fuzzy' matching process. First, the internal links are extracted from the new Wikipedia article and a map between the links and their anchor texts is created. Multiple links that share the same anchor texts are discarded. Then, each content cell in T_o is looked up in the map. If a content cell text matches any anchor text, the link is selected for that cell.

In some cases, no Wikipedia articles can be found to contain the original table, usually because the article has been deleted. In this case, the original table is kept as-is. Original Web-tables are also kept as-is, since no provenance has been recorded for them.

Thus after re-creating all tables in these datasets, they are re-annotated according to Freebase to create the named entity annotation ground truth. Each internal link in a table is firstly searched using the MediaWiki API[13] to find the corresponding Wikipedia page number. The page number is then queried on Freebase using MQL[14] to find the corresponding Freebase topic id. The end outcome of this process is a collection of tables whose content cells are annotated by Freebase topic ids.

Annotating columns and relations - Limaye200 To create the ground truth for evaluating column classification and relation enumeration, a random set of 200 tables are drawn from the LimayeAll dataset. These tables are firstly examined to identify subject columns, then manually annotated following a similar process as Venetis et al. [30]. Specifically, TableMiner and the baselines (Section 11.3) are ran on these tables and the candidate concepts ($\mathcal{C}^{\mathcal{T}}$) for all table columns and relations ($\mathcal{R}^{\mathcal{T}}$) between the subject column and other columns in each table are concatenated and presented to annotators. The annotators mark each label as *best*, *okay*, or *incorrect*. The basic principle is to prefer the most specific concept/relation 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.

11.1.2. IMDB

The IMDB dataset contains over 7,000 tables extracted from a random set of IMDB movie webpages. Each IMDB movie webpage[15] contains a table listing a 'cast' 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 entities in these columns are annotated automatically in such a way. The 'character' column is not used since they are not mapped in Freebase. The cast column is also manually labeled with *best* and *okay* concepts in the same way as for Limaye200. No subject column or relations are annotated because only one column is considered in this dataset.

11.1.3. MusicBrainz

The MusicBrainz dataset contains some 1,400 tables extracted from a random set of MusicBrainz record label webpages. Each MusicBrainz record label webpage[16] contains a table listing the music released by a production company. Each webpage uses pagination to separate very large tables, and only the first page is downloaded. The table typically has 8 columns, of which one lists music release titles (subject column) and one lists music artists. Each release title or artist has a MusicBrainz id, which can be mapped to a Freebase topic id in the same way as for the IMDB dataset. Thus entities in these two columns are annotated in such a way. Then the table columns and binary relations between the subject column and others are also annotated manually following the same procedure as for IMDB and Limaye200.

11.1.4. Dataset statistics

Table 3 shows general statistics of the datasets. Figure 4 shows the distribution of rows, columns containing named entity annotations in the ground truth, and text length of content cell by number of tokens. Tables in MusicBrainz contain no more than 50 rows

[12] Very large tables in the original datasets are split into smaller ones. The criteria of splitting is unknown. In this case, tables from the original dataset are used.

[13] http://www.mediawiki.org/wiki/API:Main_page

[14] http://www.freebase.com/query

[15] E.g., http://www.imdb.com/title/tt0071562/

[16] E.g., http://musicbrainz.org/label/9e6b4d7f49584db78504-d89e315836af

Table 3

Statistics of the datasets for evaluation. 'All' under 'Labeled columns' shows the number of both labeled NE- and literal-columns, while 'NE' refers to only NE-columns. Likewise 'All' under 'Labeled relations' shows the number of labeled relations between subject columns and either NE- or literal columns, while 'NE' refers to only relations with NE-columns.

Dataset	Tables	Subject column	Entities	Labeled columns		Labeled relations	
				All	NE	All	NE
Limaye200	200	✓	-	615	415	361	204
LimayeAll	6,310		227,046	-	-	-	-
IMDB	7,416		92,321	7,416	7,416	-	-
MusicBrainz	1,406	✓	93,266	9,842	4,218	7,030	5,624

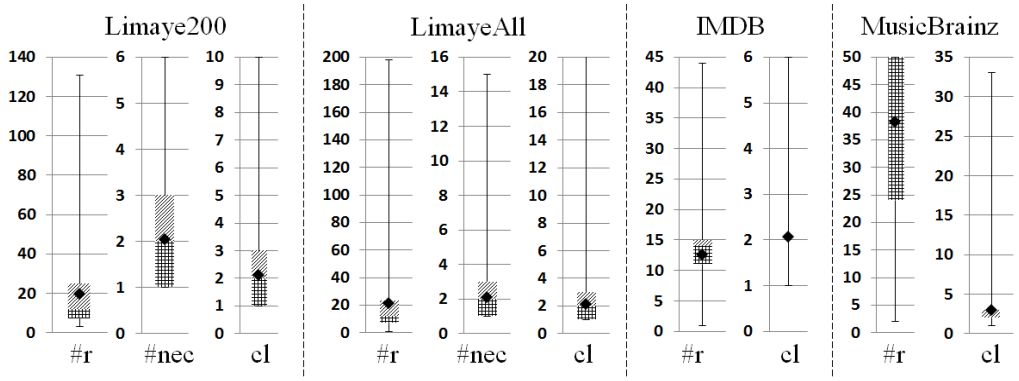

Fig. 4. Dataset statistics (*min.*, *max.* *avg.* (black diamonds), k^{th} *quantile*): *#r* - number of rows; *#nec* - number of columns containing annotated named entities in ground truth; *cl* - content cell text length in terms of number of tokens delimited by non-alphanumeric characters.

Table 4

Comparison against datasets used by state-of-the-art. The work by Muñoz et al. (2013) only applies to Wikipedia tables and cannot be generalized; the work by Wang et al. (2012) reported *in-vivo* evaluation only. '-' indicates unknown or not clear.

Method	Dataset	Tables	Columns	Entities	Relations
TableMiner	Limaye200	200	615	-	361
	LimayeAll	6,310	-	227,046	-
	IMDB	7,416	7,416	92,321	-
	MusicBrainz	1,406	9,842	93,266	-
Hignette et al. [13,14]	1 dataset	-	81	-	-
Limaye et al. [19]	4 datasets	6,310	747	142,737	90
Syed et al. [29]	1 dataset	5	21	171	-
Venetis et al. [30]	1 dataset	168	-	-	-
Buche et al. [1]	1 dataset	90	81	-	316
Mulwad et al. [23]	1 dataset	203-490	-	-	-

because of the pagination on the webpage and only the first page is downloaded. The IMDB named entity ground truth has only 1 column in every table; while for MusicBrainz, this is either 1 or 2 columns. Very long content cell text is found to be rare in LimayeAll, but slightly more frequent in MusicBrainz due to long names of classic music titles. Content cells in IMDB tables are typically person names and usually contains 1 or 2 tokens.

It is arguable that the LimayeAll and Limaye200 should be considered as the most representative datasets among the four, since they cover a significantly larger number of domains and more diverse table structures and schemata, whereas IMDB and MusicBrainz each contains only one table structure and schema. This is

particularly true when evaluating column classification and relation enumeration.

Table 4 compares against experimental datasets used by other studies. Arguably, TableMiner is evaluated using the most comprehensive collection of datasets known to date[17].

11.2. Evaluation metrics

Effectiveness Subject column detection is evaluated by Precision. Then the three subtasks of Table Interpretation are evaluated using the standard Precision, Recall and F1 measures. Since TableMiner ranks candidates by scores, only the top ranked prediction by TableMiner is considered. 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 $\frac{score}{\#top-ranked}$. 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. From a knowledge base population point of view, false-positives causes incorrect triples to be populated into knowledge bases and the LOD cloud.

For column classification and relation enumeration, evaluation reports results under both *strict* and *tolerant* mode. To evaluate column classification, the strict mode only considered *best* annotations; while the tolerant mode considers both *best* and *okay* annotations. Evaluating relation enumeration under the strict mode only considers relations between subject column and other columns in a table, and only *best* annotations are included. Under the tolerant mode, in additional to also including *okay* annotations, if TableMiner predicts correct relations between non-subject columns or the reversed relation between the subject column and other columns, then each prediction is awarded a score of 0.5.

Moreover, since most state-of-the-art has focused on only NE-columns, this work reports results obtained on NE-columns as well as both NE- and literal-columns for column classification and relation enumeration.

[17] The datasets more than doubled those used in the previous work currently under review.

Efficiency of TableMiner is assessed by empirical CPU time and savings in terms of candidate named entities need to be considered for disambiguation. As discussed before, retrieving candidate named entities, their data and constructing feature space consumes the large majority of computation. Reducing the candidate space can effectively improve Table Interpretation efficiency.

11.3. Baseline and configuration

TableMiner is evaluated against four baseline methods. Results are also compared against figures reported by state-of-the-art.

11.3.1. Baseline settings

Each baseline consists of the same *subject column detection* component, but alters *column interpretation* and *relation enumeration and literal-column annotation* (Figure 2). Baseline **name match** B_{nm} begins by disambiguating every content cell in an NE-column by firstly retrieving candidate named entities from Freebase using the text content in the cell as query, then selecting a single named entity: the highest ranked candidate whose name matches exactly the cell text. If no candidates are found to match, the top-ranked candidate is chosen. Freebase adopts a ranking algorithm that reflects both the relevance and 'popularity' of a topic in the knowledge base.

Then named entities from each content cell cast a vote for the concepts they are associated to, and the the one receiving the most votes is chosen as the label for the column. Relation enumeration follows a similar procedure. Candidate relations on each row is derived and their scores computed in the same way as TableMiner (Section 10, Equation 20). Then each candidate with a score greater than 0 is selected from the row and considered as a candidate relation for the two columns and casts a vote towards the candidate. The best relations for the two columns are those receiving the most votes.

Baselines **similarity-based** also begin by disambiguating every content cell in an NE-column, then derives column classification labels based on the disambiguated named entities from every cell. For *disambiguation*, candidate retrieval follows the same procedure as TableMiner. Then the score of a candidate named entity is the sum of a simple context score and the string similarity between the name of the candidate and the content cell text that the candidate represents. The context score is computed using Equation

10, where x is the *content row* only. Using content row as features for named entity cell disambiguation has been found common in Syed et al. [29], Limaye et al. [19] and Mulwad et al. [23]. The use of string similarity metrics is also common practice.

Candidate concepts for an NE-column are gathered based on the highest scoring candidate named entities in its containing cells (similar to Equation 14). The final score of a candidate concept depends on two components: (1) the number of highest scoring candidate named entities associated with the concept normalized by the number of rows in the table, and (2) a string similarity score between the concept label and the header text. Relation interpretation uses the same procedure from B_{nm}.

Three string similarity metrics are used to create three baselines: **Cosine** (B_{cos}), **Dice** (B_{dice}) coefficient, and **Levenshtein** (B_{lev}).

In all baselines, literal-columns are annotated the same way as TableMiner (Section 10.2). The key differences between the four baselines and TableMiner are: (1) TableMiner uses out-table context while the baselines do not; (2) TableMiner adopts a bootstrapping, incremental learning pattern with an mutually recursive optimization process to enforce interdependence between table components, the baselines however, uses an exhaustive learning strategy and is based on very simple component interdependence. Note that although the baselines do not directly replicate state-of-the-art, they follow standard practice and are composed of typical techniques, hence they are considered to be reasonable references.

11.3.2. TableMiner configuration

The convergence threshold t in the *I-Inf* algorithm is set to 0.01 in both subject column detection and column interpretation. The Web search API used for computing ws scores in subject column detection is Bing Search API[18] and default parameters are used. TableMiner uses various context in subject column detection and the three subtasks of interpretation. The weights of context in different tasks are defined in Table 5.

Context is assigned a weight of 1.0 if it is considered 'very important' or 0.5 otherwise (subjectively decided). For example, in subject column detection, if header words are found in the *webpage title* or *table captions*, they are stronger indicators (hence 'very important') of subject column than if found in *paragraphs* that are distant from tables. For classification and relation enumeration, different types of context are equally weighted , because the bag-of-words representations for candidate concept and relation are based on their labels - semantically very important features but can be very small sized therefore any matches are considered a strong signal.

In the four datasets, *semantic markups* are only available in IMDB and MusicBrainz in the Microdata format. Any23[19] is used to extract these annotations as RDF triples and the objects of triples are concatenated to create features. Annotations within *<table>* tags are excluded.

Table 6
Subject column detection results in Precision

Dataset	Tables	Precision
Limaye200	200	95.5
MusicBrainz	1406	92.7

11.4. Results and discussion

11.4.1. Subject column detection

Table 6 shows the precision of predicting subject columns in the Limaye200 and MusicBrainz datasets. The unsupervised subject column detection method achieves a precision of near 96% and 93% on the Limaye200 and MusicBrainz datasets respectively, compared to the reportedly 94-96% precision by a supervised model in Venetis et al. [30], and therein 83% by a baseline that chooses the first column not containing numeric data from left.

Figure 5 shows the convergence statistics in computing the ws score wrapped by the *I-Inf* algorithm. The number of tables that require[20] the computation of ws is 145 (73%) in Limaye200, 4,711 (75%) in LimayeAll, and 1,402 (near 100%) in MusicBrainz. Table 7 shows the statistics of the slowest convergence on each dataset.

Using Limaye200 as an example, Figure 5 suggests that to compute the ws score in subject column detection, on average, only 4 rows are processed, and in 75% of tables no more than 5 rows are processed. When measured as fraction against total numbers of rows in a table, on average, less than 35% of table rows are processed and in 75% of the dataset, no more than

[18] http://datamarket.azure.com/dataset/bing/search

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

[20] In cases that only one NE-column is available in a table, there is no need of running the subject column detection algorithm as this column is simply selected.

Table 5
Different context weight used in different tasks.

Task	In-table context			Out-table context			
	column header	row content	column content	webpage title	table caption /title	paragraphs	semantic markups
Sub.Col. detection	-	-	-	1.0	1.0	0.5	-
Entity scoring	1.0	1.0	0.5	1.0	1.0	0.5	1.0
Concept scoring	1.0	-	1.0	1.0	1.0	1.0	1.0
Relation scoring	1.0	-	-	-	-	1.0	1.0

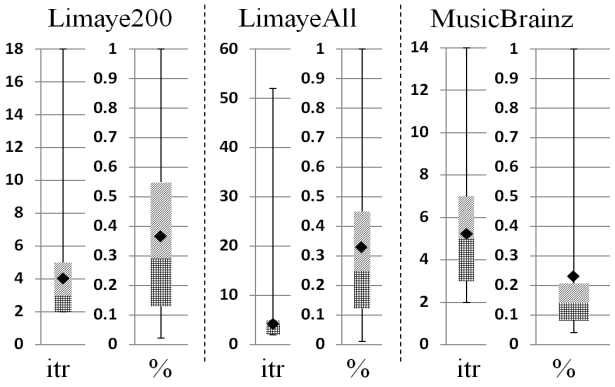

Fig. 5. Convergence statistics (max, min, average (black diamond), k^{th} quantiles) for *I-Inf* at computing ws for subject column detection. *itr* - number of iterations (rows processed) until convergence; % - fraction of table rows processed.

Table 7
Statistics of the slowest convergence in the computation of ws for subject column detection.

	Limaye200	LimayeAll	MusicBrainz
100% of rows processed in			
% of tables	6.9%	5.9%	6.5%
Avg. rows	7.2	6.5	4.6
Rows >20	0	4	0
>50% of rows processed in			
% of tables	26.2%	19.8%	12.6%
Avg. rows	6.0	6.0	5.2
Rows >20	0	7	0

55% of table rows are processed. Table 7 suggests that among all 145 tables needing computing ws, only 10 tables (equates to 6.9%) have all of their rows processed. They have an average of 7.2 rows and none has $>$ 20 rows. Then 26.2% of these 145 tables have at least 50% of their rows processed. However, on average they have only 6 rows and none of them have greater than 20 rows. All these figures suggest significant potential savings over an exhaustive approach that computes the ws score using all rows in tables. The figures on the LimayeAll and MusicBrainz datasets suggest even greater savings.

Manual examination shows that in most cases the errors fall under several categories. The first is due to duplicate values in subject columns. An extreme example is the named entity disambiguation table discussed before (i.e., 'List of peaks named Bear Mountain'), in which the subject column contains only a single unique value. The second is caused by long-named entities in subject columns. For example, the subject column in the MusicBrainz tables lists titles of music releases, some of which are very long-named consisting of more than 10 tokens. This severely penalizes their cm and ws scores as it is unlikely to find exact match of their names in table context and search result documents. In such cases, the column listing corresponding artists is favored as it contains much shorter names. The third category includes arguable (few) tables that in strict terms, do not necessarily have a subject column. These are typically tables listing events, such as lap records of racing car drivers in a particular tournament. In these cases, annotators typically selected the leftmost NE-column.

Overall the results are considered very encouraging. On the one hand, the unsupervised subject column detection method obtains comparable results to a supervised state-of-the-art method. On the other hand, by wrapping the computation of the ws feature with *I-Inf*, the method is very scalable as only a very small fraction of table content is processed in the most expensive computation of the subject column detection algorithm.

11.4.2. Effectiveness

Table 8 compares TableMiner against baselines in the disambiguation task, Table 9 shows results on column classification and Table 10 shows results on relation enumeration. The highest figures are shown in **bold**. Overall, it is clear that TableMiner always obtains the best performance in all tasks and on all

Table 8
Disambiguation results (F1) on four datasets.

	Limaye200	LimayeAll	IMDB	MusicBrainz
B_{nm}	78.3	82.2	92.8	57.0
B_{lev}	78	82.1	93.5	84.83
B_{cos}	79.2	82.7	93.4	84.76
B_{dice}	79.9	82.8	93.5	84.84
$TableMiner$	**82.3**	**83.7**	**97.6**	**84.87**

Table 9
Classification results (F1) on three datasets.

			B_{nm}	B_{lev}	B_{cos}	B_{dice}	$TableMiner$
Limaye200	NE-columns only	strict	23.9	45.4	43.5	48.9	**65.8**
		tolerant	48.8	63.8	62.2	66.1	**75.0**
	All columns	strict	31.4	48.7	46.9	51.3	**64.0**
		tolerant	51.1	63.3	61.8	65.0	**71.5**
IMDB	NE-/All columns	strict	22.7	32.3	31.9	32.0	**97.0**
		tolerant	60.0	64.6	64.4	64.5	**97.2**
MusicBrainz	NE-columns only	strict	59.1	82.6	83.4	83.9	**85.2**
		tolerant	61.7	82.6	83.6	83.9	**85.9**
	All columns	strict	54.6	72.2	72.6	72.9	**74.3**
		tolerant	55.9	72.2	72.7	72.9	**74.7**

Table 10
Relation enumeration results (F1) on two datasets.

			B_{nm}	B_{lev}	B_{cos}	B_{dice}	$TableMiner$
Limaye200	NE-columns only	strict	61.3	61.7	61.8	61.8	**72.5**
		tolerant	65.8	66.3	66.5	66.4	**76.0**
	All columns	strict	59.7	61.0	60.6	61.1	**66.2**
		tolerant	63.0	64.4	64.1	64.4	**68.7**
MusicBrainz	NE-columns only	strict	63.8	67.2	67.1	67.1	**67.9**
		tolerant	65.0	68.3	68.2	68.2	**69.1**
	All columns	strict	78.7	82.0	81.9	81.9	**82.6**
		tolerant	79.2	82.5	82.4	82.4	**83.1**

datasets. It also shows stronger advantage in the classification and relation enumeration tasks.

For **disambiguation**, even the simplistic B_{nm} obtains surprisingly competitive results on the LimayeAll and IMDB datasets. In fact, on the original datasets by Limaye et al. [19] B_{nm} obtains a surprisingly high F1 of 92.6%, significantly higher than the reported weighted average[21] of 84.1. As discussed earlier, analyses - shown in Appendix A - suggest that the datasets are sparse and less balanced, which motivated the reconstruction of the datasets.

Its highly competitive performance on the IMDB dataset could be explained by the domain and the method for ranking search results by Freebase. Movie is a highly popular domain representing a fair large proportion of Freebase. Since Freebase Search API promotes popular topics, when a person name is searched it is more likely to obtain movie-related topics as top results than any other domains. Hence by selecting the top result B_{nm} is very likely to succeed. This also explains the good performance by other base-

[21] Macro-average over all datasets taking into account the size of each dataset.

lines that consider string similarity between the content cell text and candidate entity names.

While although music is also a highly popular domain, B_{nm} could not replicate similar performance. Manual inspection shows that a fair proportion of music titles and artist uses very ambiguous names (e.g., 'Trouble', a musical release, 'Pine', an artist) shared by different topics. In contrast, other baselines perform significantly better by considering row context in tables.

Compared to the baselines, TableMiner consistently obtains the best performance. On the most representative dataset LimayeAll, it improves F1 by between 0.9 and 1.5 points. On the MusicBrainz dataset, it makes little difference from B_{lev}, B_{cos}, and B_{dice}. By examining the data it is found that the out-table context on MusicBrainz webpages are very sparse. In most cases, the webpage contains only the table. Microdata annotations are also predominantly found within tables, which are excluded from TableMiner. On the contrary, the IMDB dataset is completely the opposite: the webpages contain much richer out-table context (including pre-defined Microdata annotations), but little in-table context as the tables have only two columns and neither has column headers. TableMiner achieves significant improvement (between 3.9 and 4.6) over baselines on IMDB. This strongly suggests that out-table context serves as useful clues for disambiguating entities in table content cells, particularly when in-table context is absent. Further, the Microdata-annotations extracted from these webpages could have been a strong contributor considering that the only difference in out-table context used on LimayeAll and IMDB is that the latter also uses them as features.

For **classification**, TableMiner in most cases outperforms baselines by a very large margin. Experiments on the most representative dataset Limaye200 see an improvement of between 16.9 and 41.9 under strict mode and 8.9-26.2 under tolerant mode when only NE-columns are considered. When all columns are included, the figures are 12.7-32.6 strict and 6.5-20.4 tolerant. MusicBrainz reveals the smallest improvement of the minimum of 1.3 strict and 2.0 (both *v.s.* B_{dice}) when only NE-columns are considered, or 1.4 strict and 1.8 tolerant when all columns are included. This is due to the same reason behind TableMiner's moderate performance in the disambiguation task on this dataset: the out-table context is very sparse, thus TableMiner in most cases only uses in-table context. Again the most striking improvement is noted on the IMDB dataset. With the lack of in-table context and particularly column headers that are typically considered a crucial feature in labeling table columns, all baselines perform poorly compared against TableMiner. TableMiner more than tripled their performance under the strict mode and significantly outperforms under the tolerant mode, achieving near-perfect accuracy in both cases.

Also note that the performance by B_{nm} is generally inferior on any dataset. This is due to its inability to promote a single top ranked candidate concept in most cases, or in other words, multiple highest scoring candidates are kept for a table column and this is penalized by the scoring method. This is a severe limitation of the method as from knowledge base population point of view, considerable false positive triples will be generated. Other baselines improve this by also considering the string similarity between candidate concept's label and table column headers.

It has been noted that a typical feature adopted in the state-of-the-art to help avoid such problems is using a concept hierarchy defined within knowledge bases. Usually, concepts that are more specific (at the lower levels of the hierarchy) are given higher weights [19,23], which work as tie-breakers. However, concept hierarchies are not necessarily available in all knowledge bases, such as Freebase. Nevertheless, TableMiner is able to predict a single best concept candidate in most cases without such knowledge.

Results on **relation enumeration** follow similar patterns. On the multi-domain Limaye200 dataset, an improvement of between 10.7-11.2 strict and 9.5-10.2 tolerant is noted when only relations between NE-columns are considered. Regardless of column types, the figures are 5.1-6.5 strict and 4.3-5.7 tolerant. Improvement on MusicBrainz is smaller: the minimum of 2.8 strict and 0.8 tolerant with NE-columns only, and the minimum of 0.6 strict and 0.7 tolerant if all columns are included. Again this could be attributed to the lack of out-table context in this dataset.

To compare against **state-of-the-art**, Table 11 assembles reported results of several related studies and TableMiner's results on the Limaye200 and LimayeAll datasets (which are also used by others). Since state-of-the-art only tackles NE-columns, TableMiner's results in the NE-column only setting are used. Due to the use of different datasets and knowledge bases, the figures may not be directly comparable. Nevertheless, they provide useful references. In general, TableMiner obtains very competitive results. Note that Mulwad et al. [23] use 3-7% of the original Limaye datasets only, and the disambiguation module requires training data

Table 11
TableMiner on Limaye200 (classification and relation enumeration) and LimayeAll (disambiguation) and state-of-the-art reported figures. All results are based on NE-columns only. [23] and [30] report results under *tolerant* mode only, weigted average are used where necessary.

	Disamb.	Class.	Relation
TableMiner *strict*	83.7	65.8	72.5
TableMiner *tolerant*		75.0	76.0
Limaye et al. [19]	84.1	44.5	58.3
Mulwad et al. [23]	90.9	54.9	83.5
Venetis et al. [30]	-	68.6	54.8

while TableMiner is completely unsupervised. Also, on the original Limaye datasets (LimayeAll-Original), TableMiner achieves an F1 of 92%(see Appendix A).

Overall, results of these experiments are very encouraging. The rich context model adopted by TableMiner - especially the usage of out-table context - enables TableMiner to achieve the best performance in all tasks, and significantly outperforming baseline models that ignore out-table contextual features in most cases. In particular, column classification appears to benefit most, suggesting that out-table context provides very useful clues for labeling table columns with semantic concepts. The superior performance noticed on the IMDB dataset double-confirms this, and also shows that existing semantic markups within webpages can be very useful features in this task. Intuitively, when describing table content we tend to focus on the general information rather than specific data in individual table components, which possibly explains the particular contribution by out-table context to the column classification task. Moreover, results on the IMDB dataset also suggest that TableMiner can be easily adapted to solve tasks in list structures, which are essentially single column tables without headers.

11.4.3. Efficiency

The efficiency of TableMiner is compared against the baselines B_{lev}, B_{cos} and B_{dice} that represent the exhaustive strategy. The three baselines are almost identical in terms of efficiency since they only differ in the string similarity metric used. Therefore, only B_{lev} is compared as an example.

Table 12 compares the CPU hours of TableMiner against B_{lev} on the four datasets, and shows the proportion of time spent by TableMiner on 'data retrieval' - including searching for candidate named entities and retrieving their triple data from Freebase or local cache (which is used wherever possible to avoid sending repeated queries to Freebase). All systems were tested on the same computer platform and no parallelization was used. TableMiner is shown to be faster than B_{lev}, despite its complicated feature modeling and algorithmic computation.

Table 12
CPU time (hours) observed for TableMiner as savings against the baseline B_{lev}.

	Savings (CPU hours)	Savings (% of B_{lev} CPU time)	% of CPU time on data retrieval
Limaye200	3.7	21.7%	99%
LimayeAll	45.7	15.2%	97%
IMDB	3.9	4.1%	96%
MusicBrainz	34.3	28.6%	97%

While Limaye et al. [19] suggest that data retrieval and feature construction accounts for up to 99% of CPU time, Table 12 shows that in this experiment, data retrieval alone has already caused the majority of time. Based on these observations, an effective way to improve efficiency is to reduce named entity disambiguation candidates, therefore, cutting down both the number data retrieval operations and feature construction operations.

TableMiner improves efficiency in two ways. First, the one-sense-per-discourse hypothesis ensures that values repeating on multiple content cells in non-NE columns are disambiguated collectively costing only one operation. This avoids both repeated data retrieval and feature construction operations for the same set of named entity candidates. Whereas classic methods without this hypothesis disambiguate these cells individually, costing extra computation. Second, the bootstrapping approach in TableMiner reduces the total number of candidate named entities by firstly creating initial column classification using *partial* instead of *entire* column content, then using this outcome to limit candidate space in named entity disambiguation. Table 13 compares the total number of candidate named entities processed during disambiguation operations in TableMiner against (1) the exhaustive baseline B_{lev}, and (2) a *would-be exhaustive TableMiner* (**exh-TableMiner**) which exploits one-sense-per-discourse but firstly disambiguates every content cell in the column before computing column classification (i.e., without using *I-Inf* to create initial classification to constrain disambiguation). TableMiner's improvement is shown as reduction in % against the two reference methods.

Compared against the exhaustive baseline B_{lev}, TableMiner reduces the total number of candidate

Table 13
Candidate named entity reduction in disambiguation operations by TableMiner compared against exhaustive baseline B_{lev} and a *would-be exhaustive TableMiner (exh-TableMiner)*.

	B_{lev}	exh-TableMiner	
	Overall	Overall	Constrained Disambiguation phase
Limaye200	48.4%	30.4%	39.2%
LimayeAll	52.2%	32.7%	41.4%
IMDB	10.6%	10.6%	59.4%
MusicBrainz	66.4%	44.3%	50.4%

named entities to be disambiguated by 10-67%. Note that the smallest improvement on the IMDB dataset is due to (1) the dataset being dominated by very small tables (see Figure 4 on average < 15 rows), and as a result, *I-Inf* in the *LEARNING* phase does not converge or converges relatively slowly (to be discussed in Section 11.5.2); and (2) one-sense-per-discourse being void since only one column (hence the subject column) is available. Empirically, this translates to the very small improvement in CPU time shown in Table 12.

The reduction narrows when compared against *exh-TableMiner*, however it still represents a substantial improvement. If only the content cells at *constrained disambiguation* of the *LEARNING* phase are considered, TableMiner improves over *exh-TableMiner* by a significant 39-60%.

Furthermore, as mentioned before, one potential issue that may damage TableMiner's efficiency is that during the *UPDATE* phase, new named entity candidates can be retrieved and processed, due to the change of winning candidate concepts on a column in each update iteration. In the worst case, the total number of candidates to be retrieved from Freebase equals that in an exhaustive method. However, empirically it is found that this rarely happens. The number of named entity candidates to be retrieved from Freebase in the *UPDATE* phase are shown in Table 14, compared to the total number of named entity candidates retrieved from Freebase by TableMiner in all phases. Further, Table 15 shows the number of iterations until stabilization is reached in the *UPDATE* phase. It suggests that the *UPDATE* phase stabilizes very fast. Compared to the semantic message passing algorithm in Mulwad et al. [23], at high-level, the iterative *UPDATE* phase is similar to running semantic message passing on a graph containing two types of variable nodes - column headers and content cells - and one type of factor nodes that model compatibility between the candidates of the

Table 14
Number of named entity candidates need to be retrieved from Freebase at the *UPDATE* phase.

	Number	‰of total
Limaye200	177	3.9‰
LimayeAll	7,762	5.1‰
IMDB	418	0.8‰
MusicBrainz	867	1.6‰

Table 15
Number of iterations until stabilization at the *UPDATE* phase. *1 itr* - fraction of tables on which the *UPDATE* phase stabilizes after 1 iteration.

	Max.	Avg.	1 itr.
Limaye200	4	1.3	72.8%
LimayeAll	5	1.3	75.8%
IMDB	4	1.2	79.7%
MusicBrainz	4	1.3	73.1%

variable nodes. This is a much simpler graph than that used by Mulwad et al., which can fail to converge and a threshold must be used to exit the loop.

11.5. The effect of using partial table content for interpretation

To specifically evaluate the effect of inference using *partial* as opposed to entire table column content, four 'slim' versions of TableMiner are created. Firstly, the *UPDATE* phase is dropped from the column interpretation component in order to separate its potential effect on learning accuracy. This creates TM^{nu}_{iinf} (nu means 'no update'), which begins with *sample based classification* using *I-Inf*. The classification results based on the partial column content are considered to be the final and used in *constrained disambiguation*. Then, three alternative models are created by replacing the automatically determined *I-Inf* stopping criteria with arbitrarily set sample size. The first uses a maximum of 10 rows as sample to create column classification (TM^{nu}_{10}), the second uses a maximum of 20 rows (TM^{nu}_{20}), and the third uses the entire column (TM^{nu}_{all}) and is equivalent to *exh-TableMiner* without *UPDATE*. TM^{nu}_{10} and TM^{nu}_{20} can be considered as supervised versions of TableMiner without *UPDATE* as ideally, the best threshold is to be empirically

derived[22]. Results of these settings are also compared against the best performing baseline B_{dice} and the full TableMiner.

11.5.1. Accuracy

Tables 16, 17 and 18 show F1 accuracy obtained on the disambiguation, classification and relation enumeration tasks respectively. First and foremost, TM^{nu}_{iinf} outperforms the best performing baseline in almost all occasions except two cases: disambiguation on MusicBrainz and classification on MusicBrainz under 'NE-column only' and 'strict' mode, in which cases the difference is very small (0.2 and 0.6 point). Without the *UPDATE* phase, the key differences of TM^{nu}_{iinf} from the baseline are the use of out-table context as features and using partial table column content for column classification. Therefore, the consistent improvement over the baseline is another strong confirmation of the benefits of using out-table context in semantic Table Interpretation, particularly in the task of column classification where the improvement is the greatest.

Comparison against other slim versions of TableMiner shows that TM^{nu}_{iinf} is very competitive: it is able to obtain the best performance in many cases and where it does not, it achieves close-to-best performance (with a difference of merely 0.1-0.6 point). In particular, in the classification task on the Limaye200 dataset, TM^{nu}_{iinf} outperforms all the other versions under any settings. Considering that Limaye200 is the most representative dataset for this task while IMDB and MusicBrainz each has only one type of table schema, the results suggest that the *I-Inf* algorithm is very much capable of automatically selecting optimal sample size for column classification.

Also interesting to note is that the exhaustive version TM^{nu}_{all} appears to have little advantage over any sample-based versions. It only outperforms all the rest in 5 cases, where the improvement is merely 0.1-0.2 point. In many cases, results are even worse than sample-based versions. This could be attributed to the addition of noisy candidate concepts when more named entity content cells are disambiguated compared to sample-based versions.

Compared against the full TableMiner, the results show that the addition of the *UPDATE* phase indeed further improves learning accuracy, particularly in the classification and relation enumeration tasks where the benefits are substantial in most cases.

[22]Zwicklbauer et al. [39] have shown that a sample size between 10 and 20 rows lead to close-to-maximum performance in column classification.

Table 19
Statistics of the slowest convergence of *I-Inf* when used for *sample based classification* in the *LEARNING* phase.

	Limaye 200	Limaye All	IMDB	Music Brainz
100% of rows processed in				
% of tables	33.8%	30.7%	27.6%	7.5%
Avg. rows	11.0	10.0	13.0	8.5
Rows >20	10	190	0	24
>50% of rows processed in				
% of tables	47.8%	47.6%	49.2%	13.9%
Avg. rows	11.4	11.0	12.0	8.6
Rows >20	16	442	0	47

11.5.2. I-Inf convergence speed

While the *I-Inf* algorithm is already shown to be very competitive in ensuring learning accuracy, it is also very efficient. Figure 6 shows the convergence statistics in the *LEARNING* phase of TableMiner on all datasets. Table 19 shows the statistics of the slowest convergence on each dataset.

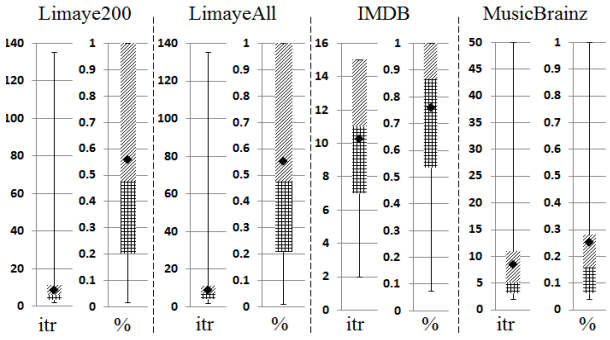

Fig. 6. Convergence statistics (max, min, average (black diamond), k^{th} quantiles) for *I-Inf* at the *LEARNING* phase. *itr* - number of iterations (rows processed) until convergence; % - fraction of content cells processed in a column

On all datasets, *I-Inf* in the *LEARNING* phase typically converges in an average of 10 iterations. In other words, only 10 content cells are processed to create initial column classification. This also explains the observation that the learning accuracy obtained by TM^{nu}_{iinf} is very similar to TM^{nu}_{10}. It represents about 55% of data in an average table from the Limaye200 and LimayeAll datasets, less than 30% in the case of MusicBrainz, and about 75% for IMDB where the majority (75%) are small tables (see Figure 4). In the cases

Table 16

Named entity disambiguation results (F1) of the 'slim' versions of TableMiner compared against the best baseline B_{dice} and the full TableMiner. The highest figures among all slim versions of TableMiner are highlighted in **bold**.

	Limaye200	LimayeAll	IMDB	MusicBrainz
B_{dice}	79.9	82.8	93.5	84.84
$TableMiner$	82.3	83.7	97.6	84.87
TM_{10}^{nu}	81.3	**83.3**	**96.4**	84.69
TM_{20}^{nu}	**81.4**	**83.3**	96.1	84.82
TM_{all}^{nu}	81.3	**83.3**	96.1	**84.85**
TM_{iinf}^{nu}	81.2	**83.3**	**96.4**	84.62

Table 17

Classification results (F1) of the 'slim' versions of TableMiner compared against the best baseline B_{dice} and the full TableMiner. The highest figures among all slim versions of TableMiner are highlighted in **bold**.

			B_{dice}	$TableMiner$	TM_{10}^{nu}	TM_{20}^{nu}	TM_{all}^{nu}	TM_{iinf}^{nu}
Limaye200	NE-columns only	strict	48.9	65.8	56.5	56.5	56.5	**56.9**
		tolerant	66.1	75.0	71.7	71.8	71.6	**72.1**
	All columns	strict	51.3	64.0	56.2	56.4	56.4	**56.7**
		tolerant	65.0	71.5	68.6	68.8	68.6	**69.1**
IMDB	NE-/All columns	strict	32.0	97.0	**65.1**	60.3	60.3	64.5
		tolerant	64.5	97.2	**81.7**	79.3	79.3	81.4
MusicBrainz	NE-columns only	strict	83.9	85.2	83.3	**83.4**	**83.4**	83.3
		tolerant	83.9	85.9	85.2	**85.4**	**85.4**	85.2
	All columns	strict	72.9	74.3	73.4	73.5	**73.6**	73.4
		tolerant	72.9	74.7	74.4	**74.5**	**74.5**	74.4

Table 18

Relation enumeration results (F1) of the 'slim' versions of TableMiner compared against the best baseline B_{dice} and the full TableMiner. The highest figures among all slim versions of TableMiner are highlighted in **bold**.

			B_{dice}	$TableMiner$	TM_{10}^{nu}	TM_{20}^{nu}	TM_{all}^{nu}	TM_{iinf}^{nu}
Limaye200	NE-columns only	strict	61.8	72.5	70.0	**70.2**	69.6	69.6
		tolerant	66.4	76.0	74.9	**75.1**	74.6	74.6
	All columns	strict	61.1	66.2	63.8	**64.2**	63.9	63.9
		tolerant	64.4	68.7	67.5	**67.9**	67.6	67.6
MusicBrainz	NE-columns only	strict	67.1	67.9	67.5	**67.6**	67.6	67.5
		tolerant	68.2	69.1	68.6	**68.7**	68.7	68.6
	All columns	strict	81.9	82.6	82.1	**82.2**	**82.2**	**82.2**
		tolerant	82.4	83.1	82.6	**82.7**	**82.7**	82.6

that TableMiner does not converge or converges slowly (Table 19), the tables are very small.

11.5.3. Overall

Combining the discusses above, the *I-Inf* algorithm has been shown to be both effective and efficient. First, compared against TM_{all}^{nu} a representation of an exhaustive method in the classification task, it significantly reduces computation yet outperforms TM_{all}^{nu} in many cases. Particularly on the Limaye200 dataset, it only process about 55% of table content yet producing more accurate column labels than TM_{all}^{nu}. Second, compared against the supervised versions TM_{10}^{nu} and TM_{20}^{nu}, it obtains either the best or close-to-best performance, but eliminates the need for learning the sam-

ple size threshold and avoids over-fitting by automatically determining optimal sample sizes. Third, when compared against baselines, TM^{nu}_{iinf} consistently outperforms; strongly confirming the usefulness of out-table context as well as the benefits of using partial data in the task of semantic Table Interpretation.

12. Conclusion

This article introduced TableMiner, a novel semantic Table Interpretation method that annotates tabular data for semantic indexing, search and knowledge base population. TableMiner advances state-of-the-art in several ways. First, its bootstrapping learning method enables it to make interpretation based on partial table content hence solves the task in a more efficient way than classic methods that exhaustively processes the entire table. Second, embedded in its efficient learning algorithm is a novel method for automatically ranking and selecting sample data to seed learning. Third, TableMiner is the first to use various context outside tables as features in the task. In particular, it is the first system using pre-defined semantic markups within webpages to deal with NLP tasks. Fourth, it offers a comprehensive solution, solving all subtasks of Table Interpretation and deals with both named entity and literal content cells. And finally, releasing the largest collection of datasets for the task.

TableMiner is also completely unsupervised, and highly generic. The features used by TableMiner are present in any knowledge bases used by state-of-the-art Table Interpretation methods. In general, it can be used with any Linked Data sets.

Extensive experiments show that TableMiner outperforms all baselines on any datasets under any settings. On the two most representative datasets covering multiple domains and diverse table schemata, it significantly improves the baselines (up to 42 points). When compared against state-of-the-art on similar datasets based on reported figures, it also obtains better performance. Compared against classic, exhaustive methods, TableMiner reduces empirical CPU time by up to 29% and in the column classification task alone, uses only 55% of table content (as opposed to 100% by exhaustive methods) to label columns on the two most representative datsets. These results confirm the the initial motivation of this work, that it is possible to create Table Interpretation methods by using various out-table context, it is possible to create effective and efficient Table Interpretation methods by exploiting various out-table context and using only partial table content.

TableMiner is however, still limited in a number of ways. First, relation enumeration is yet incomplete, as TableMiner only handles binary relations between the subject column and other columns. Second, the ranking method in the sample selection is very simplistic, alternative methods may be investigated. Likewise, alternative measures for convergence in the *I-Inf* may be explored. Finally, TableMiner is evaluated using Freebase. Ideally, it should be evaluated using other knowledge bases as well. These issues will be explored in future work.

References

[1] Patrice Buche, Juliette Dibie-Barthélemy, Liliana Ibanescu, and Lydie Soler. Fuzzy web data tables integration guided by an ontological and terminological resource. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):805–819, 2013.

[2] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of VLDB Endowment*, 1(1):538–549, August 2008.

[3] Michael J. Cafarella and Eugene Wu. Uncovering the relational web. In *Proceedings of the 11th International Workshop on Web and Databases*, 2008.

[4] Fabio Ciravegna, Anna Lisa Gentile, and Ziqi Zhang. Lodie: Linked open data for web-scale information extraction. In Diana Maynard, Marieke van Erp, and Brian Davis, editors, *SWAIE*, volume 925 of *CEUR Workshop Proceedings*, pages 11–22. CEUR-WS.org, 2012.

[5] Silviu Cucerzan. 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)*, pages 708–716, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[6] Li Ding, Dominic DiFranzo, Alvaro Graves, James R. Michaelis, Xian Li, Deborah L. McGuinness, and James A. Hendler. Twc data-gov corpus: Incrementally generating linked government data from data.gov. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1383–1386, New York, NY, USA, 2010. ACM.

[7] William A. Gale, Kenneth W. Church, and David Yarowsky. One sense per discourse. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 233–237, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[8] Anna Lisa Gentile, Ziqi Zhang, Isabelle Augenstein, and Fabio Ciravegna. Unsupervised wrapper induction using linked data. In *Proceedings of the seventh international conference on Knowledge capture*, K-CAP '13, New York, NY, USA, 2013. ACM.

[9] Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of the 11th Confer-*

ence of the European Chapter of the Association for Computational Linguistics (EACL 2006), Trento, Italy, April 2006.

[10] Wael H. Gomaa and Aly A. Fahmy. Article: A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, April 2013. Published by Foundation of Computer Science, New York, USA.

[11] Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs, and Anupam Joshi. Rdf123: From spreadsheets to rdf. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pages 451–466, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, COLING '92, pages 539–545, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[13] Gaëlle Hignette, Patrice Buche, Juliette Dibie-Barthélemy, and Ollivier Haemmerlé. An ontology-driven annotation of data tables. In *Proceedings of the 2007 international conference on Web information systems engineering*, WISE'07, pages 29–40, Berlin, Heidelberg, 2007. Springer-Verlag.

[14] Gaëlle Hignette, Patrice Buche, Juliette Dibie-Barthélemy, and Ollivier Haemmerlé. Fuzzy annotation of web data tables driven by a domain ontology. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pages 638–653, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] Jun'ichi Kazama and Kentaro Torisawa. Exploiting wikipedia as external knowledge for named entity recognition. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 698–707, 2007.

[16] Vijay Krishnan and Christopher D. Manning. 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*, ACL-44, pages 1121–1128, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[17] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proc. IJCAI-97*, 1997.

[18] Andreas Langegger and Wolfram Wöß. Xlwrap — querying and integrating arbitrary spreadsheets with sparql. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 359–374, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.

[20] Chunliang Lu, Lidong Bing, Wai Lam, Ki Chan, and Yuan Gu. Web entity detection for semi-structured text data records with unlabeled data. *International Journal of Computational Linguistics and Applications*, 2013.

[21] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. Triplifying wikipedia's tables. In Anna Lisa Gentile, Ziqi Zhang, Claudia d'Amato, and Heiko Paulheim, editors, *The Linked Data for IE workshop at ISWC2013*, volume 1057 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[22] Varish Mulwad, Tim Finin, and Anupam Joshi. Automatically generating government linked data from tables. In *Working notes of AAAI Fall Symposium on Open Government Knowledge: AI Opportunities and Challenges*, November 2011.

[23] Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic message passing for generating linked data from tables. In *International Semantic Web Conference (1)*, Lecture Notes in Computer Science, pages 363–378. Springer, 2013.

[24] Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi. T2ld: Interpreting and representing tables as linked data. In Axel Polleres and Huajun Chen, editors, *ISWC Posters and Demos*, CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[25] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, January 2007. Publisher: John Benjamins Publishing Company.

[26] Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, S?ren Auer, Juan Sequeda, and Ahmed Ezzat. A survey of current approaches for mapping of relational databases to rdf, 01 2009.

[27] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *In Advances in Neural Information Processing Systems 17*, pages 1185–1192, 2004.

[28] Zareen Syed, Tim Finin, and Anupam Joshi. Wikipedia as an ontology for describing documents. In *Proceedings of the Second International Conference on Weblogs and Social Media*. AAAI Press, March 2008.

[29] Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. Exploiting a web of semantic data for interpreting tables. In *Proceedings of the Second Web Science Conference*, April 2010.

[30] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proceedings of VLDB Endowment*, 4(9):528–538, June 2011.

[31] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. Understanding tables on the web. In *Proceedings of the 31st international conference on Conceptual Modeling*, ER'12, pages 141–155, Berlin, Heidelberg, 2012. Springer-Verlag.

[32] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. Probase: a probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 481–492, New York, NY, USA, 2012. ACM.

[33] R. Zanibbi, D. Blostein, and J.R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. *International Journal of Document Analysis and Recognition*, 7:1–16, 2003.

[34] Ziqi Zhang. Named entity recognition: Challenges in document annotation, gazetteer construction and disambiguation, 2013.

[35] Ziqi Zhang. Towards effective and efficient semantic table interpretation. In *Under review: ISWC2014*, 2014.

[36] Ziqi Zhang, Anna Lisa Gentile, and Fabio Ciravegna. Recent advances in methods of lexical semantic relatedness Äì a survey. *Natural Language Engineering*, FirstView:1–69, 4 2012.

[37] Ziqi Zhang and José Iria. A novel approach to automatic gazetteer generation using wikipedia. In *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, People's Web '09, pages 1–9, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[38] GuoDong Zhou, Jian Su, Jie Zhang, and Min Zhang. Exploring various knowledge in relation extraction. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 427–434, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[39] Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. Towards disambiguating web tables. In *International Semantic Web Conference (Posters & Demos)*, pages 205–208, 2013.

Appendix

A. Testing with the original Limaye datasets

As discussed before, to evaluate named entity disambiguation of TableMiner, the original Limaye dataset [19] was re-created by (1) downloading the most recent versions of corresponding webpages; (2) re-annotate named entity content cells in ground truth according to Freebase (LimayeAll). In addition, another version of the ground truth has also been created by only re-annotating named entity content cells without re-downloading the most recent webpages. To contrast against the new Limaye dataset, this is to be called the original Limaye dataset, **LimayeAll-Original**. TableMiner has also been tested on LimayeAll-Original. However, results show that the simplistic baseline B_{nm} obtains surprisingly the best performance. Analysis of LimayeAll-Original suggests that it is less balanced than LimayeAll and hence could be biased. This section discusses this part of experiment.

To create LimayeAll-Original, the original collection of webpages and ground truth are kept as-is. Then each Wikipedia internal link in the table ground truth is mapped to a Freebase topic id using the MediaWiki API and the Freebase using MQL API, following the procedures discussed before in Section 11.1.1.

TableMiner and the four baselines are tested on this dataset for named entity disambiguation and results are shown in Table 20. Surprisingly, the most simplistic baseline B_{nm} obtains the highest accuracy (F1) while TableMiner scores the second. Considering the intuition behind B_{nm} this could suggest that LimayeAll-Original is biased towards popular named entities. To obtain a better understanding, two types of analysis have been carried out.

Table 20
Disambiguation results (F1) on LimayeAll-Original.

B_{nm}	B_{lev}	B_{cos}	B_{dice}	$TableMiner$
92.6	91.2	91.8	91.5	92.0

First, the dataset statistics of LimayeAll-Original is gathered and compared against LimayeAll, as shown in Table 21. The statistics show that LimayeAll nearly doubled LimayeAll-Original in terms of the number of named entity annotations in the ground truth. Further, LimayeAll also has a larger population of 'short' entity names, as measured by the number of tokens in named entity content cells. Typically, short names are much more ambiguous than longer names, thus making disambiguation tasks more challenging. Together this could have made LimayeAll a more balanced dataset with much improved level of diversity, increasing the difficulty of task and possibly offsetting the bias in LimayeAll-Original.

Table 21
Comparing the statistics of the re-constructed LimayeAll dataset against the original LimayeAll-Original.

	LimayeAll	LimayeAll-Original
Avg.# rows	21.2	20.8
Avg.# NE-annotated cols.	2.1	1.1
Total # annotated NE cells (*A.N.C.*)	227,046	118,927
# single-token A.N.C.	30.5%	24.5%
# A.N.C. with two tokens	44.4%	45%

Second, to obtain a more balanced view of the performance of different systems on LimayeAll-Original, the results created by the baselines and TableMiner are manually inspected and re-annotated. To do so, a random set of 100 tables are selected from LimayeAll-Original and the output by B_{nm}, B_{lev} and TableMiner are collected. The output of B_{cos} and B_{dice} are not examined since they only differ from B_{lev} in terms of the string similarity metric and the performance of the three systems is little different. Then for each method, the predicted named entity annotations that are already covered by the ground truth in LimayeAll-Original are excluded. Then, in the remaining annotations, those that all three systems predict the same are removed. The remainder are the ones that the three systems 'disagree' on, and are manually validated[23]. This resulted in a total of 903 entity annotations, of which 579 is predicted correctly by at least one system. Table 22 shows the precision by the three systems based on this part of data. TableMiner significantly outperforms the two baselines. Manual inspection of 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 75.1, 35.7 higher than B_{nm} and 32.4 higher than B_{lev}.

In summary it is believed that these analyses further confirms the superiority of TableMiner against the baselines.

[23] Included in: http://staffwww.dcs.shef.ac.uk/people/Z.Zhang/resources/tableminer/data.tar.gz

Table 22

Precision on the manually annotated 903 named entity annotations that the three systems disagree on.

	Precision	Precision either-or (579)
B_{nm}	25.3	39.4
B_{lev}	32.1	42.7
TableMiner	48.2	75.1