Table Search, Generation and Completion

by

Shuo Zhang

A dissertation submitted in fulfilment of
the requirements for the degree of
PHILOSOPHIAE DOCTOR (PhD)

University of
Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science
2019

University of Stavanger
N-4036 Stavanger
NORWAY
`www.uis.no`

ISBN 978-82-7644-873-3
ISSN 1890-1387

PhD Thesis UiS no. 478

Acknowledgement

Pursuing a Ph.D. was a sudden decision I made by flipping a coin when I was finishing my master studies. To this day, I still have mixed feelings about that decision–to start on a completely new subject. The resulting lack of confidence and experience in computer science contributed to my uncertainty of how to handle the ups and downs of doctoral life properly. If I knew then the challenges that lay before me, I would surely have said no to this opportunity. Nevertheless, in the end I have to admit it has been worthwhile. This endeavour has been a groping forward, it has been a spiritual fast, and it has taught me when to stop struggling and when to push forward. The experience has equipped me with a new creed of life:

Wherever you find yourself, accept life as it is:
Enjoy the peaks, enjoy the lows, and enjoy stagnation.

I was like a blank page at the beginning, without much background in computer science. Today you can easily see this page is filled with the words and colors imparted to me by my supervisor Prof. Dr. Krisztian Balog. He taught me without reservation all the skills I needed to conduct research, including developing ideas, writing papers, delivering presentations, to name a few. I also want to acknowledge Krisztian's real effect on my life beyond a merely formal description. Sometimes I teased him for being slow and steady as a nineteenth-century steam train, while I must be a runaway horse in his eyes. But I firmly believe that differing voices bring you closer to the truth, and I appreciate from my heart all the debates we had. In the end, "iron shapes iron".

Since last autumn, I was able to take an internship at Bloomberg L.P. and research visit at Carnegie Mellon University in the USA. My four-month internship at Bloomberg was a great experience, and I met many excellent researchers there. I would like to thank Edgar Meij, as well as my other colleagues there, including Ridho Reinanda, Miles Osborne, Giorgio Stefanoni, Oana, Antonia, Nimesh, and all the rest, for giving me such a warm welcome and much help. I was supervised by Prof. Dr. Jamie Callan when I visited CMU. I am grateful to Jamie for generously sharing his experience and wisdom, about both papers and academic careers. I would also thank Zhuyun Dai for her good company while I was at CMU. I would also like to extend my thanks to my co-supervisor, Prof. Dr. Heri Ramampiaro from NTNU, for kindly hosting our visits in Trondheim, and the head of our department, Dr. Tom Ryen, for always being supportive.

I am grateful and honored to have Prof. Dr. Brian Davison, Prof. Dr. Min Zhang, and Prof. Dr. Hein Meling serving as my committee members. They reviewed my dissertation and provided me with excellent suggestions.

IAI witnesses our growth. I would like to thank the people who were in this team in the past years. Dario Gigliotti, Faegheh Hasibi, Heng Ding, Jan R. Benetka and Trond Linjordet. It is lucky to know someone to share similar views with in this adventure.

Thanks to all the local friends in Stavanger. We did so many crazy things together, which will be my long lasting memories. Most importantly, I convey my heartfelt appreciation to my family. Thanks for all the support, always giving me the freedom to follow my heart and be a wayward kid.

Lastly, I want to thank myself. You are such a lucky guy to get so much love from the others. Thanks for your tireless running and being kind. If you can, please continue.

Contents

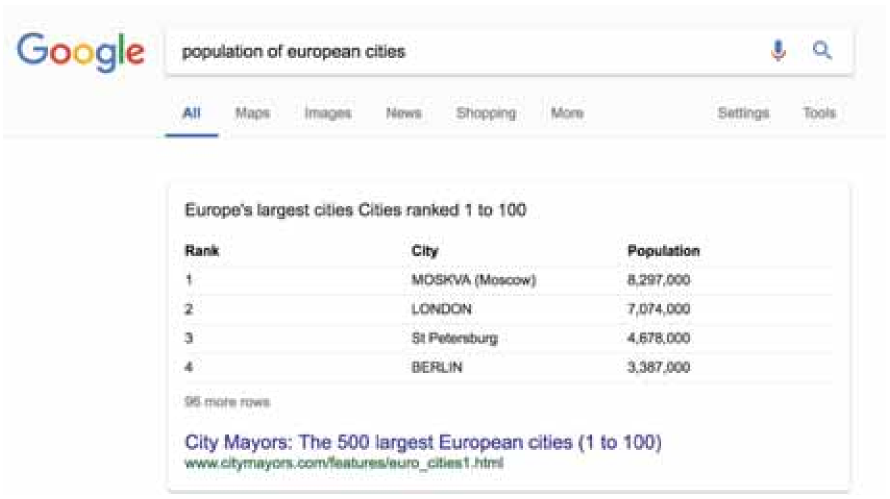
Chapter 1

Introduction

Tables are one of those "universal tools" that are practical and useful in many application scenarios. Tables can be used to collect and organize information from multiple sources and then turn that information into knowledge (and, ultimately, support decision-making) by performing various operations, like sorting, filtering, and joins. Because of this, a large number of tables exist already out there on the Web, which represent a vast and rich source of structured information that could be utilized.

The focus of the thesis is on developing methods for assisting the user in completing a complex task by providing intelligent assistance for working with tables. Specifically, our interest is in *relational tables*, which describe a set of entities along with their attributes.

Imagine the scenario that a user is working with a table, and has already entered some data in the table. Intelligent assistance can include providing recommendations for the empty table cells, searching for similar tables that can serve as a blueprint, or even generating automatically the entire a table that the user needs. The table-making task can thus be simplified into just a few button clicks. Motivated by the above scenario, we propose a set of novel tasks such as table search, table generation, and table completion. *Table search* is the task of returning a ranked list of tables in response to a query. Google, for instance, can now provide tables as direct answers to plenty of queries, especially when users are searching for a list of things. Figure 1.1 shows an example. *Table generation* is about automatically organizing entities and their attributes in a tabular format to facilitate a better overview. *Table completion* is concerned with the task of augmenting the input table with additional tabular data. Figure 1.2 illustrates a scenario that recommends row and column headings to populate the table with and automatically completes table values from verifiable sources. In this thesis, we propose methods and evaluation resources for addressing these tasks.



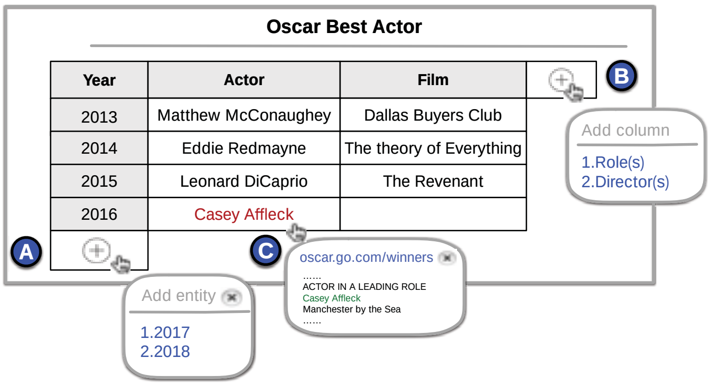
Europe's largest cities Cities ranked 1 to 100

Rank	City	Population
1	MOSKVA (Moscow)	8,297,000
2	LONDON	7,074,000
3	St Petersburg	4,678,000
4	BERLIN	3,387,000

96 more rows

City Mayors: The 500 largest European cities (1 to 100)
www.citymayors.com/features/euro_cities1.html

Figure 1.1: Google search results for the query "population of european cities". It retrieves an existing table as the direct answer.



Oscar Best Actor

Year	Actor	Film
2013	Matthew McConaughey	Dallas Buyers Club
2014	Eddie Redmayne	The theory of Everything
2015	Leonard DiCaprio	The Revenant
2016	Casey Affleck	

Figure 1.2: Assistance with table completion by populating rows (A) and columns (B), as well as automatically completing table values from verifiable sources (C).

1.1 Research Questions

The research questions of this thesis center around the way to equip tables with intelligent assistance functionalities such as searching, generating, and completing tables. To address these, we propose a set of tasks and methods. Below, we detail the main research questions along with the proposed tasks.

Table search is an important task on its own and is regarded as a fundamental step in many other table mining and extraction tasks as well, like table integration or data completion. Yet, it has not received due attention, and especially not from an information retrieval perspective. Table search functionality is also available in commercial products; e.g., Microsoft Power Query provides smart assistance features based on table search. Depending on the type of the query, table search may be classified as *keyword query search* and *table query search*. However, public test collections and proper evaluation methodology are lacking, in addition to the need for better ranking techniques. Our first question deals with keyword table search. Traditional keyword table search relies on lexical matching between the keyword query and tabular text. We seek to answer the following main research question.

> **RQ 1.** Can we move beyond lexical matching and improve keyword table search performance by incorporating semantic matching?

For table-related tasks, the table that the user is working on can also be taken as the query. Table search using a table as query boils down to computing the similarity between the query table and candidate table. We thus ask:

> **RQ 2.** Can we develop an effective and theoretically sound table matching method for measuring and combining table element level similarity, without resorting to hand-crafted features?

There is a growing body of research on assisting users in the labor-intensive process of table creation by helping them to augment tables with data [Yakout et al., 2012, Ahmadov et al., 2015a]. However, there are still some tasks that have not been tackled in these flavors, like table population. As such, we introduce and address the task of populating the input tables with additional rows and columns, and seek to answer:

> **RQ 3.** How to populate table rows and column heading labels?

Organizing results, that is, entities and their attributes, in a tabular format facilitates a better overview. There exist two main families of methods that can return a table as answer to a keyword query by: (i) performing table search to find existing tables on the Web [Cafarella et al., 2008a, 2009, Venetis et al., 2011, Pimplikar and Sarawagi, 2012, Nguyen et al., 2015], or (ii) assembling a table in a row-by-row fashion [Yang et al., 2014] or by joining columns from multiple tables [Pimplikar and Sarawagi, 2012]. However, these methods are limited to returning tables that already exist in their entirety or at least partially (as complete rows/columns). Another line of work aims to translate a keyword or natural language query to a structured query language (e.g., SPARQL), which can be executed over a knowledge base [Yahya et al., 2012]. While in principle these techniques could return a list of tuples as the answer, in practice, they are targeted for factoid questions or at most a single attribute per answer entity. More importantly, they require data to be available in a clean, structured form in a consolidated knowledge base. Motivated by these, we ask the following research question:

> **RQ 4.** How to generate an output table as response to a free text query?

The last task of fact finding is related to both table completion and generation. In particular, table completion tasks focus on the core column entities and headings, and it can not provide suggestions for the rest table cells. Assisting to fill in tabular cells with evidence can complement the table generation by verifying the value by tracking back to its source of origin. As such, we seek to answer:

> **RQ 5.** How to find facts with supporting evidence when completing tables?

1.2 Main Contributions

We now summarize the main contributions of this thesis.

1. **Synthesizing existing research on table-related tasks:** We provide a comprehensive survey on the broader topic of web table mining, retrieval and augmentation, which also includes the work that has been done outside information retrieval (i.e., databases, data mining and the Semantic Web).
2. **Revisiting the task of keyword table search, establishing test collections, baselines and methods:** We introduce and formalize the ad hoc table ranking task, and present both unsupervised and supervised baseline approaches. We present a set of novel semantic matching methods that go beyond lexical similarity. We develop a standard test collection for this task and demonstrate the effectiveness of our approaches.
3. **Defining the task of query-by-table, developing test collections and methods:** We introduce the query-by-table paradigm, adapt existing methods, and present a discriminative approach that combines handcrafted features from the literature. We develop a general a table matching framework and specific instantiations of this framework. We construct a purpose-built test collection, perform a thorough experimental evaluation, and provide valuable insights and analysis.
4. **Proposing two table completion tasks, evaluation measures, test collectins and method:** We introduce and formalize two specific tasks for providing intelligent assistance with tables: row population and column population. We present generative probabilistic methods for both tasks, which combine existing approaches from the literature with

novel components. We design evaluation methodology and develop a process that simulates a user through the process of populating a table with data. We perform an experimental evaluation and carry out a detailed analysis of performance.

5. **Introducing the table generation task, presenting methods to generate tables on the fly.** We introduce the task of on-the-fly table generation and propose an iterative table generation algorithm. We develop feature-based approaches for core column entity ranking and schema determination, and design an entity-oriented fact catalog for fast and effective value lookup. We perform extensive evaluation on the component level and provide further insights and analysis.
6. **Defining the task of value finding with supporting evidences, developing test collections, establishing baselines and methods:** We present the CellAutoComplete framework for finding cell values in relational tables, which consists of preprocessing, candidate value finding, and value ranking steps. Specific novel technical contributions include the heading-to-heading and heading-to-predicate matching components and as well as the features designed for combining evidence from multiple sources and for predicting empty values. We develop a purpose-built test collection based on Wikipedia tables and perform an extensive experimental evaluation. Our experiments show that our LTR approach substantially outperforms existing data augmentation techniques. We show that our CellAutoComplete framework can also be used for checking existing values in tables.
7. **SmartTable demo:** The main contributions of this demonstrator are twofold. First, we integrate the above assistance functionality into an online spreadsheet application. Second, we describe the task-specific indexing structures employed, and evaluate the efficiency of our implementation in terms of response time. SmartTable is implemented using a HTML5 front-end and a Python+ElasticSearch back-end. It uses DBpedia as the underlying knowledge base and a corpus of 1.6M tables extracted from Wikipedia.

1.3 Organization of the Thesis

This section introduces how the thesis is organized. See Fig 1.3 for an illustration. The thesis starts with the introduction and background chapters (Chapter 1-2), followed by five technical chapters, each dedicated to a specific table-related task (Chapter 3-7), and concludes in Chapter 8. We assume that the reader is familiar with traditional information retrieval approaches and evaluation methods. For an introduction to these, see, e.g., [Zhai and Massung, 2016].

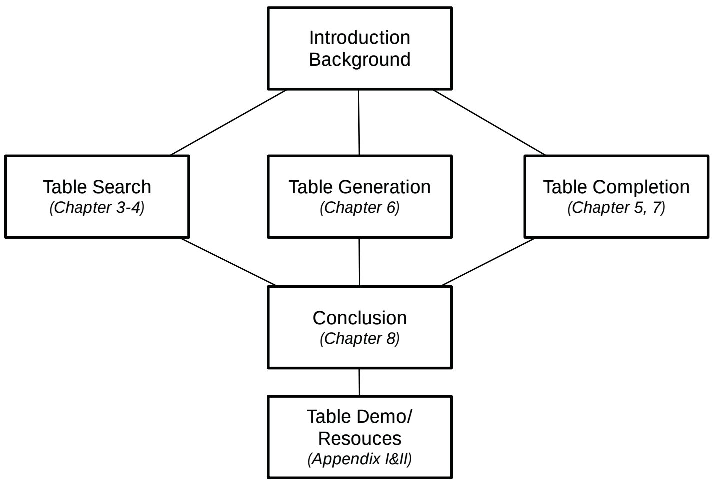


Figure 1.3: Overview of thesis organization.

Chapters 1-2 are dedicated to the introduction and background for the rest of the thesis. Chapter 1 introduces the motivation, research questions, contributions, and the origins of the thesis. Chapter 2 describes the table-related tasks and existing approaches.

Chapter 3 defines and addresses the problem of *table retrieval for keyword query*: answering a keyword query with a ranked list of tables.

Chapter 4 proposes a novel task called *query-by-table*: given an input table, return a ranked list of tables. It boils down to computing the similarity between the input table and a candidate table, which is referred as *table matching*.

Chapter 5 introduces two specific tasks: populating rows with additional instances (entities) and populating columns with new headings. It investigates methods for utilizing different table elements for these tasks.

Chapter 6 introduces and addresses the task of the *on-the-fly table generation*: given a query, generate a relational table that contains relevant entities (as rows) along with their key properties (as columns) by combining fragmentation information from web tables or from knowledge bases.

Chapter 7 addresses the task of finding table cell values with supporting evidence, to support users in the labor-intensive process of creating relational tables.

Chapter 8 concludes with remarks and provides future research directions.

Appendix A details the deployment of the SmartTable demonstrator, which has two assistance functionalities of row and column population.

Appendix B lists the collection of resources developed in this thesis, which are made publicly available.

1.4 Origins

The content of this thesis is based on a number of papers. Some of these have been published, while others are currently under review. We list the publications that are directly relevant to this thesis below.

P1. Shuo Zhang. **SmartTable: Equipping Spreadsheets with Intelligent Assistance Functionalities**, in Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '18), pages 1447-1447, 2018.
[Integrated in Chapter 1].

P2. Shuo Zhang and Krisztian Balog. **Web Table Extraction, Retrieval and Augmentation: A Survey,** accepted by ACM Transactions on Intelligent Systems and Technology (TIST) (subject to minor revisions).
[Related to contribution 1; included in Chapter 2].

P3. Shuo Zhang and Krisztian Balog. **Ad Hoc Table Retrieval using Semantic Similarity**, in Proceedings of the Web Conference 2018 (WWW '18), pages 1553-1562, 2018.
[Related to RQ1 and contribution 2; included in Chapter 3].

P4. Shuo Zhang and Krisztian Balog. **Recommending Related Tables**, arXiv preprint arXiv:1907.03595, 2019.
[Related to RQ2 and contribution 3; included in Chapter 4].

P5. Shuo Zhang and Krisztian Balog. **Semantic Table Retrieval using Keyword and Table Queries,** submitted to a journal, under review.
[Related to RQ1, RQ2, contribution 2, and contribution 3; included in Capters 3-4].

P6. Shuo Zhang and Krisztian Balog. **EntiTables: Smart Assistance for Entity-Focused Tables**, in Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17), pages 255-264, 2017.
[Related to RQ3 and contribution 4; included in Chapter 5].

P7. Shuo Zhang and Krisztian Balog. **On-the-fly Table Generation**, in Proceedings of the 41st International ACM SIGIR Conference on Research

and Development in Information Retrieval (SIGIR '18), pages 595-604, 2018.
[Related to RQ4 and contribution 5; included in Chapter 6].

P8. Shuo Zhang and Krisztian Balog. **Auto-completion for Data Cells in Relational Tables**, In Proceedings of The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), 2019.
[Related to RQ5 and contribution 6; included in Chapter 7].

P9. Shuo Zhang, Vugar Abdul Zada, and Krisztian Balog. **SmartTable: A Spreadsheet Program with Intelligent Assistance**, in Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '18), pages 1297-1300, 2018.
[Related to contribution 7; included in Appendix A].

The following papers are not directly related to this thesis but brought insights to working with semi-structured data and capturing semantics.

P10. Shuo Zhang and Krisztian Balog. **Design Patterns for Fusion-Based Object Retrieval**, in Proceedings of the 39th European Conference on Information Retrieval (ECIR '17), pages 684-690, 2017.

P11. Faegheh Hasibi, Dario Garigliotti, Shuo Zhang, and Krisztian Balog. **Supervised Ranking of Triples for Type-Like Relations-The Cress Triple Score at WSDM Cup**, in WSDM Cup 2017.

P12. Faegheh Hasibi, Krisztian Balog, Dario Garigliotti, and Shuo Zhang. **Nordlys: a Toolkit for Entity-oriented and Semantic Search**, in Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17), pages 255-264, 2017.

P13. Heng Ding, Shuo Zhang, Dario Garigliotti, and Krisztian Balog. **Generating High-Quality Query Suggestion Candidates for Task-Based Search**, in Proceedings of the 40th European Conference on IR Research (ECIR '18), pages 625-631, 2018.

P14. Li Deng, Shuo Zhang, and Krisztian Balog. **Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval**, in Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), pages 1029-1032, 2019.

P15. Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. **Novel Entity Discovery from Web Tables for Knowledge Base Population**, under review.

P16. Shuo Zhang, Jamie Callan, and Krisztian Balog. **Generating New Wikipedia Categories from Tabular Data**, under review.

P17. Shuo Zhang and Krisztian Balog. **Web Table Extraction, Retrieval, and Augmentation** (tutorial overview), in Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), pages 1409-1410, 2019.

Chapter 2

Background

In this chapter, we present existing research on web tables. First, we discuss the types of tables and public corpora (Sect. 2.1). Then, we organize existing literature into six main categories of information access tasks: table extraction (Sect. 2.2), table interpretation (Sect. 2.3), table search (Sect. 2.4), question answering on tables (Sect. 2.5), knowledge base augmentation (Sect. 2.6) and table augmentation (Sect. 2.7). The relationship between the different tasks is shown in Fig. 2.1. For each of these tasks, we identify and describe seminal approaches, present relevant resources, and point out interdependencies among the different tasks.

2.1 Table Types and Corpora

In this section, we formally introduce tables (Sect. 2.1.1), present various types of tables (Sect. 2.1.2), and provide an overview of publicly available datasets (Sect. 2.1.3).

2.1.1 The Anatomy of a Table

A table T is grid of cells arranged in rows and columns. Tables are used as visual communication patterns and as data arrangement and organization tools. In this chapter, our primary focus is on *web tables*, that is, tables embedded in webpages. Below, we define elements of a web table. We refer to Fig. 2.2 for an illustration.

Table page title The table page title T_p is the title of the webpage which embeds the table T.

Table caption The caption of a table, T_c, is a short textual label summarizing what the table is about.

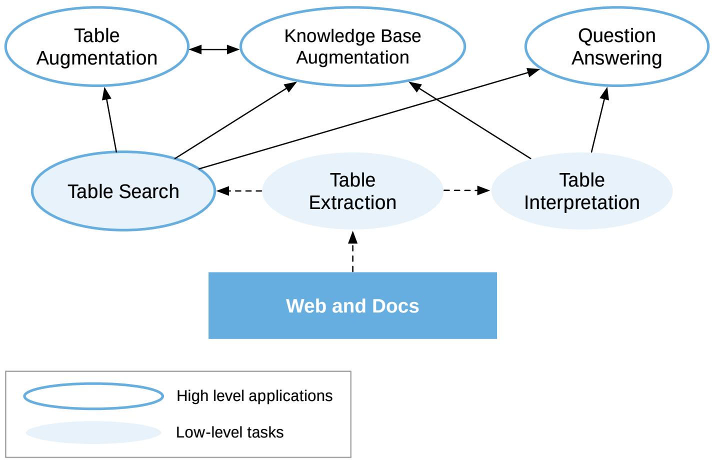
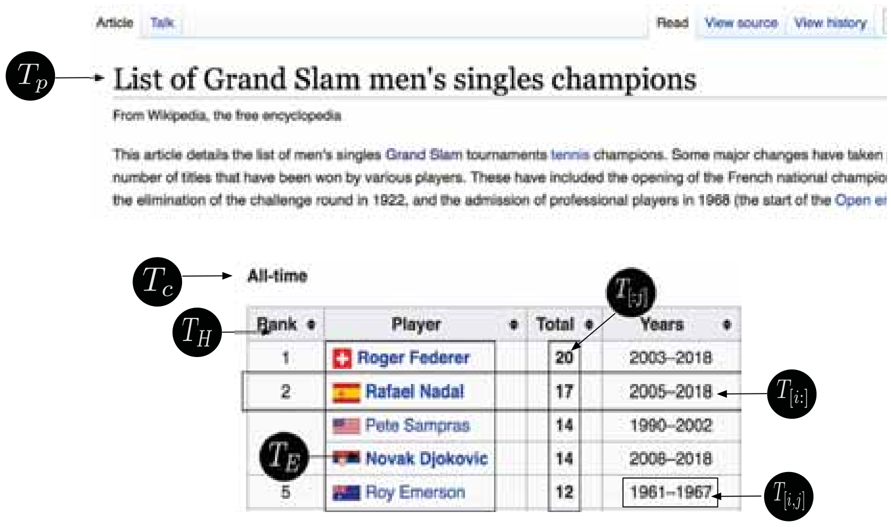


Figure 2.1: Table-related information access tasks and their relationships.



Article Talk Read View source View history

List of Grand Slam men's singles champions

From Wikipedia, the free encyclopedia

This article details the list of men's singles Grand Slam tournaments tennis champions. Some major changes have taken
number of titles that have been won by various players. These have included the opening of the French national champio
the elimination of the challenge round in 1922, and the admission of professional players in 1968 (the start of the Open e

All-time

Rank	Player	Total	Years
1	Roger Federer	20	2003–2018
2	Rafael Nadal	17	2005–2018
	Pete Sampras	14	1990–2002
	Novak Djokovic	14	2008–2018
5	Roy Emerson	12	1961–1967

Figure 2.2: Illustration of table elements in a web table: table page title (T_p), table caption (T_c), table headings (T_H), table cell ($T_{[i,j]}$), table row ($T_{[i,:]}$), table column ($T_{[:,j]}$), and table entities (T_E).

Table headings Table headings, T_H, is a list of labels defining what each table row/column is about. Headings are typically in the first row/-column in a table. In case of relational tables (see below, in Sect. 2.1.2), table headings are also referred to as *table schema* or *attribute names*.

Table cell A table cell $T_{[i,j]}$ is specified with the row index i and column index j. Table cells hold (possibly empty) values and are considered as atomic units in a table.

Table row A table row $T_{[i,:]}$ is a list of table cells lying horizontally in line i of a table.

Table column A table row $T_{[:,j]}$ is a list of table cells lying vertically in column j of a table.

Table entities Tables often mention specific entities, such as persons, organizations, locations. Table entities T_E is a set consisting of all the entities that are mentioned in the table.

2.1.2 Types of Tables

A number of table classification schemes have been proposed in the literature. We start by reviewing those, then propose a normalized categorization based on the main aspects these share.

In early work, Wang and Hu [2002a] make a distinction between genuine and non-genuine tables:

- *Genuine tables* are leaf tables, i.e., do not contain other tables, lists, forms, images or other non-text formatting tags in a cell. Furthermore, they contain multiple rows and columns.
- *Non-genuines tables* refers to those that are not leaf tables.

Cafarella et al. [2008b] classify web tables into five main categories:

- *Extremely small tables* are those having fewer than two rows or columns.
- *HTML forms* are used for aligning form fields for user input.
- *Calendars* are a specific table type, for rendering calendars.
- *Non-relational tables* are characterized by low quality data, e.g., used only for layout purposes (many blank cells, simple lists, etc.).
- *Relational tables* contain high-quality relational data.

Crestan and Pantel [2011] develop a fine-grained classification taxonomy, organized into a multi-layer hierarchy.

- *Relational knowledge tables* contain relational data.
  - *Listings* refer to tables consisting a series of entities with a single attribute. In terms of layout direction, these are further classified as *vertical listings* or *horizontal listings*.
  - *Attribute/value tables* describe a certain entity along with its attributes.

    - *Matrix tables* have the same value type for each cell at the junction of a row and a column. *Calendars*, for example, can be regarded as matrix tables.
    - *Enumeration tables* list a series of objects that have the same ontological relation (e.g., hyponomys or siblings).
    - *Form tables* are composed of input fields for the user to input or select values.
- *Layout tables* do not contain any knowledge and are used merely for layout purposes.
    - *Navigational tables* are meant for navigating within or outside a website.
    - *Formatting tables* are used for visually organizing content.

Lautert et al. [2013] refine the classification scheme of Crestan and Pantel [2011].

- *Relational knowledge tables*
    - *Horizontal tables* place attribute names on top (column header). Each column corresponds to an attribute.
    - *Vertical tables* place attribute names on the left (row header). Each row represents an attribute.
    - *Matrix tables* are three dimensional data sets, where headers are both on the top and on the left.
- *Layout tables*, as before, are subdivided into *navigational tables* and *formatting tables*.

Relational knowledge tables are further classified according to a secondary type taxonomy.

- *Concise tables* contain merged cells (cells having the same value are merged into one) to avoid value repetition.
- *Nested tables* contain a table in a cell.
- *Multivalued tables* refer to tables containing multiple values in a single cell. If all values in one cell come from one domain, they are named as *simple multivalued web tables*, if not, they are called *composed multivalued value tables*.
- *Splitted tables* present sequentially ordered repetitions in row/column headers (i.e., each label is repeated in every x cell).

With a particular focus on web spreadsheets, Chen and Cafarella [2013] define the following type taxonomy:

- *Data frame spreadsheets* contain data frames, each consisting of two regions: data (numeric values) and headings (attribute names). These are further classified based on how they are arranged:
  - *Hierarchical left spreadsheets* place attributes on the left of the data region.
  - *Hierarchical top spreadsheets* put attributes on top of the data region.
- *Non-data frame (flat) spreadsheets* do not contain a data frame.
  - *Relation spreadsheets* can be converted into the relational model [Codd, 1970].
  - *Form spreadsheets* are designed for human-computer interaction.
  - *Diagram spreadsheets* are for visualization purposes.
  - *List spreadsheets* consist of non-numeric tuples.
  - *Other spreadsheets* include schedules, syllabi, scorecards, and other files without a clear purpose.

Eberius et al. [2015] distinguish tables along two dimensions: content and layout. In terms of content, they adopt the classification scheme by Wang and Hu [2002a]. Considering layout purposes, they sort tables according to their logical structure into the following categories:

- *Horizontal listings* align cells horizontally.
- *Vertical listings* align cells vertically.
- *Matrix tables* refer to numerical tables.

Lehmberg et al. [2016] distinguish between three main types of tables:

- *Relational tables* contain a set of entities, which could exist in rows (*horizontal*) or columns (*vertical*); the remainder of the cells contain their descriptive attributes.
- *Entity tables* describe a certain entity.
- *Matrix tables* are referring to a numerical table.

The above categorization systems are quite diverse, which is not surprising considering that each was designed with a different use-case in mind. Nevertheless, we can observe two main dimensions along which tables are distinguished: content and layout. We propose a normalized classification scheme, which is presented to Table 2.1. In the remainder of this chapter, we shall follow this classification when referring to a certain type of table.

Table 2.1: Classification of table types in this chapter.

Dimension	Type	Description
Content	Relational	Describes a set of entities with their attributes
	Entity	Describes a specific entity
	Matrix	A three dimensional data set, with row and column headers
	Other	Special-purpose tables, including lists, calendars, forms, etc.
Layout	Navigational	Tables for navigational purposes
	Formatting	Tables for visual organization of elements

Table 2.2: Overview of table corpora.

Table corpora	Type	#tables	Source
WDC 2012 Web Table Corpus	Web tables	147M	Web crawl (Common Crawl)
WDC 2015 Web Table Corpus	Web tables	233M	Web crawl (Common Crawl)
Dresden Web Tables Corpus	Web tables	174M	Web crawl (Common Crawl)
WebTables	Web tables	154M	Web crawl (proprietary)
WikiTables	Wikipedia tables	1.6M	Wikipedia
TableArXiv	Scientific tables	0.34M	arxiv.org

2.1.3 Table Corpora

A number of table corpora have been developed in prior work, which are summarized in Table 2.2.

WDC Web Table Corpus

There are two versions of WDC Web Table Corpus,[1] which were released in 2012 and 2015 respectively. The 2012 version contains 147 million web tables, which were extracted from the 2012 Common Crawl corpus (consisting of 3.5 billion HTML pages). Tables in this corpus are roughly classified as relational or not relational in terms of layout. Statistically, 3.3 billion HTML pages were parsed and 11.2 billion tables were identified; tables that are not innermost (that is, contain other tables in their cells) were discarded. 1.3% of the remaining tables (originating from 101 million different web pages) were labeled as relational tables. Tables in this corpus are not classified further and neither is table context data provided.

The WDC 2015 Web Table Corpus, constructed by Lehmberg et al. [2016], contains 10.24 billion *genuine* tables. The extraction process consists of two steps: table detection and table classification. The percentages of *relational*,

[1] http://webdatacommons.org/framework/

entity, and *matrix* tables are 0.9%, 1.4%, and 0.03% respectively. The remaining 97.75% accounts for *layout* tables. When storing a table, its orientation is also detected, indicating how the attributes are placed. In horizontal tables, the attributes are placed in columns, while in vertical tables they represent rows. There are 90.26 million relational tables in total. Among those, 84.78 million are horizontal and 5.48 million are vertical. The average number of columns and rows in horizontal tables are 5.2 and 14.45. In vertical tables, these numbers are 8.44 and 3.66, respectively. Lehmberg et al. [2016] also extract the column headers and classify each table column as being numeric, string, data, link, boolean, or list. The percentages of the numeric and string columns are 51.4% and 47.3%, respectively. Besides, the text surrounding the table (before and after) is also provided.

Furthermore, Lehmberg et al. [2016] provide the English-language Relational Subset, comprising of relational tables that are classified as being in English, using a naive Bayesian language detector. The language filter considers a table's page title, table header, as well as the text surrounding the table to classify it as English or non-English. The average number of columns and rows in this subset are 5.22 and 16.06 for horizontal tables, and 8.47 and 4.47 for vertical tables. The percentages of numeric and string columns are 51.8% and 46.9%.

A total of 139 million tables in the WDC 2015 Web Table Corpus are classified as entity tables. Out of these, 76.70 million are horizontal and 62.99 million are vertical tables. The average number of columns and rows are 2.40 and 9.08 for horizontal tables, and 7.53 and 2.06 for vertical tables. The column data types are quite different from that of relational tables. String columns are the most popular, amounting to 86.7% of all columns, while numeric columns account for only 9.7%.

The complete corpus as well as the different subcorpora are made publicly available.[2]

Dresden Web Table Corpus

Eberius et al. [2015] also extracted tables from the Common Crawl web corpus. The total number of tables is 174 million, which is reduced to 125 million after filtering with regards to content-based duplication. The Dresden Web Table Corpus contains only the core table data, and not the entire HTML page. Even though the corpus is not available for download directly, the table extraction framework (extractor code and companion library for working with the data set) is made publicly available.[3]

[2]http://webdatacommons.org/webtables/#results-2015
[3]https://wwwdb.inf.tu-dresden.de/misc/dwtc/

WebTables

Cafarella et al. [2008b] extracted 154 million high-quality relational web tables from a (proprietary) general-purpose web crawl. Unfortunately, this corpus is not made public. However, frequency statistics of attributes, known as the ACSDb dataset (cf. Sec. 2.7.2), is available for download.[4]

Wikipedia Tables

Bhagavatula et al. [2015] focused on Wikipedia and extracted 1.6 million high-quality relational tables. Each table is stored as a JSON file, including table body, table caption, page title, column headers, and the number of row and columns. The existing links in the tables are also extracted and stored in a separate file. The corpus is available for download.[5] This is the collection we use in Chapters 3-7.

Scientific Tables

Scientific tables are a particular type of table, which contain valuable knowledge and are available in large quantities. The TableArXiv corpus[6] consists of 341,573 tables, extracted from physics e-prints on arxiv.org. Along with the corpus, 105 information needs and corresponding relevance judgements are also provided for the task of scientific table search.

2.2 Table Extraction

A vast number of tables can be found on the Web, produced for various purposes and storing an abundance of information. These tables are available in heterogenous format, from HTML tables embedded in webpages to files created by spreadsheet programs (e.g., Microsoft Excel). To conveniently utilize these resources, tabular data should be extracted, classified, and stored in a consistent format, resulting ultimately in a table corpus. This process is referred to as *table extraction*. In this section, we present approaches for the table extraction task, organized around three main types of tables: web tables, Wikipedia tables, and spreadsheets.

2.2.1 Web Table Extraction

Table extraction is concerned with the problem of identifying and classifying tables in webpages, which encompasses a range of more specific tasks, such as relational table classification, header detection, and table type classification. These three tasks (relational table classification, header detection, and

[4]https://web.eecs.umich.edu/~michjc/data/acsdb.html
[5]http://websail-fe.cs.northwestern.edu/TabEL/
[6]http://boston.lti.cs.cmu.edu/eager/table-arxiv/

table type classification) are commonly approached as a supervised learning problem and employ similar features; these features are summarized in Tables 2.3 and 2.4. In the follows, we organize the literature according to the three tasks.

Relational table classification

The identification of tables on the Web is usually straightforward based on HTML markup. Tables, however, are also used extensively for formatting and layout purposes. Therefore, web table extraction involves a data cleaning subtask, i.e., identifying and filtering out "bad" tables (where "bad" usually denotes non-relational tables). *Relational table classification* (also known as identifying high-quality or genuine tables) refers to the task of predicting whether a web table contains relational data.

One of the pioneering works utilizing tables on the Web is the WebTables project [Cafarella et al., 2008a,b]. Cafarella et al. [2008b] regard relational tables as high-quality tables, and filter those by training a rule-based classifier. The classifier uses table characteristics, like table size and table tags, as features. The model is trained on a set of manually annotated tables (as being relational or non-relational) by two human judges. As a result, they construct a high-quality table corpus, consisting of 154 million tables, filtered from 14.1 billion HTML tables (cf. Sect. 2.1.3). Balakrishnan et al. [2015] follow a similar approach to [Cafarella et al., 2008b] for relational table classification, but use a richer set of features, which include both syntactic and semantic information. Syntactic features are related to the structure of the table, as in [Cafarella et al., 2008b] (e.g., number of rows and columns). Semantic features are obtained by (i) determining whether the table falls into a boilerplate section of the containing page, (ii) detecting subject columns (using a binary SVM classifier trained based on one thousand manually labeled tables), (iii) identifying column types (which will be detailed later, in Sect. 2.3.1), (iv) and detecting binary relationships between columns (by analyzing how these relationships are expressed in the text surrounding the table). Wang and Hu [2002b] define a table as *genuine*, if it is a leaf table where no subtable exists in any of the cells. They employ machine learned classifiers (decision trees and support vector machines) to classify relational tables, using three main groups of features: layout features, content type features, and word group features. The layout features and most of the content features are listed in Tables 2.3 and 2.4. As for word group features, Wang and Hu [2002b] treat each table as a document and compute word frequency statistics. In follow-up work, the authors also experiment with other machine learning methods (Naive Bayes and weighted kNN), using the same set of features [Wang and Hu, 2002a]. Building on [Wang and Hu, 2002b], Eberius et al. [2015] carry out relational table classification as well as classification according to layout type (vertical listings, horizontal listing, and matrix tables). Their first method performs classification along both dimensions simultaneously, using a single layer. Their second approach separates the two tasks into two layers, where the first layer executes table detection and, subsequently, the second layer determines the layout type. Various machine learning methods are employed, including decision trees, Random Forests, and SVMs, using a combination of global and local features; a selection of features are listed in Table 2.3 and Table 2.4. As a result, Eberius et al. [2015] classify millions of tables and generate the Dresden Web Table Corpus (DWTC, cf. Sect. 2.1.3).

To obtain metadata for relational tables, Eberius et al. [2015] consider whether tables have a header row or not. They find that 71% of the tables in the corpus have a relational header. For the remaining 29%, they attempt to generate synthetic labels by comparing the column content to similar columns that have proper labels. Cafarella et al. [2009] design a system called OCTOPUS, which combines search, extraction, data cleaning, and integration. Further challenges related applying WebTables in practice, including table identification and table semantics recovery, are detailed in [Balakrishnan et al., 2015]. The resulting system, Google Fusion Tables, is made publicly available.[7]

Header detection

To extract data in a structured format, the semantics of tables need to be uncovered to some extent. One question of particular importance is whether the table contains a header row or column. This is known as the task of *header detection*. Headers may be seen as a particular kind of table metadata. Header detection is commonly addressed along with the other two tasks and uses similar features (cf. Tables 2.3 and 2.4).

Table type classification

Another type of metadata that can help to uncover table semantics is table type. *Table type classification* is the task of classifying tables according to a pre-defined type taxonomy (cf. Sect. 2.1.2 for the discussion of various classification schemes). Additional metadata extracted for tables includes the embedding page's title, the table's caption, and the text surrounding the table.

The same features that are intended for relational table classification and header detection can also be used for table type classification [Wang and Hu, 2002b,a, Lehmberg et al., 2016, Chen and Cafarella, 2013, Cafarella et al., 2008b]. For example, the features listed in Tables 2.3 and 2.4 are used in [Eberius et al., 2015] for both relational table classification and table type classification. Instead of directly classifying tables as relational or not, this

[7]https://research.google.com/tables

Table 2.3: Selected features for *relational table classification (RTC)* and *header detection (HD)* (Part 1/2).

Features	Explanation	Task	Source
Global layout features			
Max rows	Maximal number of cells per row	RTC, TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Max cols	Maximal number of cells per column	RTC, TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Max cell length	Maximal number of characters per cell	RTC, TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
#rows	Number of rows in the table	RTC, HD	[Cafarella et al., 2008b]
#cols	Number of columns in the table	RTC, HD	[Cafarella et al., 2008b]
%rows	Percentage of rows that are mostly NULL	RTC	[Cafarella et al., 2008b]
#cols non-string	Number of columns with non-string data	RTC	[Cafarella et al., 2008b]
μ	Average length of cell strings	RTC	[Cafarella et al., 2008b]
δ	Standard deviation of cell string length	RTC	[Cafarella et al., 2008b]
$\frac{\mu}{\delta}$	Cell string length	RTC	[Cafarella et al., 2008b]
%length one	Percentage of columns with $\lvert len(row_1) - \mu \rvert > 2\delta$	HD	[Cafarella et al., 2008b]
%length two	Percentage of columns with $\delta \leq \lvert len(row_1) - \mu \rvert \leq 2\delta$	HD	[Cafarella et al., 2008b]
%length three	Percentage of columns with $\lvert len(row_1) - \mu \rvert < \delta$	HD	[Cafarella et al., 2008b]
Avg rows	Average number of cells across rows	RTC, TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Avg cols	Average number of cells across columns	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Avg cell length	Average length of characters per cell	RTC,TTC	[Crestan and Pantel, 2011, Eberius et al., 2015, Wang and Hu, 2002b]

can also be done indirectly by saying that a table is relational if relational information can successfully be extracted from it [Chen and Cafarella, 2013]. Table extraction is also involved in a number of other studies, but these datasets are not publicly available. For example, with the purpose of data integration, Wang et al. [2012] use a rule-based filtering method to construct a corpus of 1.95 billion tables. For a type-classification study, Crestan and Pantel [2011] extract a corpus of 8.2 billion tables. Using a more fine-grained type taxonomy (see Sect. 2.1.2), table type classification is approached as a multi-class classification problem. Crestan and Pantel [2011] propose a rich set of features, including global layout features, layout features, and content features. Global layout features include the maximum number of rows, cols,

Table 2.4: Selected features for *relational table classification (RTC)*, *header detection (HD), and table detection (TD)* (Part 2/2).

Features	**Explanation**	**Task**	**Source**
Layout features			
Std dev rows	Standard dev. of the number of cells per row	RTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Std dev cols	Standard dev., of the number of cells per column	RTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Std dev cell length	Standard dev. of the number of characters per cell	RTC	[Crestan and Pantel, 2011, Eberius et al., 2015, Wang and Hu, 2002b]
Local length avg	Average size of cells in segment	RTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Local length variance	Variance of size of cells in segment	RTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Content features			
%body non-string	Percentage of non-string data in table body	*HD*	[Cafarella et al., 2008b]
%header non-string	Percentage of non-string data in the first row	*HD*	[Cafarella et al., 2008b]
%header punctuation	Percentage of columns with punctuation in the first row	*HD*	[Cafarella et al., 2008b]
Local span ratio	Ratio of cells with a ⟨span⟩ tag	RTC,TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Local ratio header	Cells containing ⟨th⟩ tag	RTC,TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Local ratio anchor	Cells containing ⟨a⟩ tag	RTC,TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Local ratio input	Cells containing ⟨input⟩ tag	RTC,TTC	[Crestan and Pantel, 2011, Eberius et al., 2015]
Ratio img	Ratio of cells containing images	RTC,TTC	[Crestan and Pantel, 2011, Eberius et al., 2015, Wang and Hu, 2002b]
Ratio form	Ratio of cells containing forms	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Ratio hyperlink	Ratio of cells containing hyperlinks	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Ratio alphabetic	Ratio of cells containing alphabetic characters	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Ratio digit	Ratio of cells containing numeric characters	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Ratio empty	Ratio of empty cells	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]
Ratio other	Ratio of other cells	RTC,TTC	[Eberius et al., 2015, Wang and Hu, 2002b]

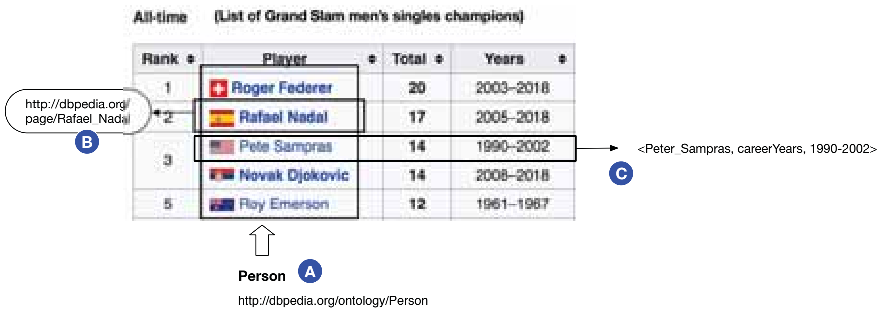
and maximum cell length. Layout features include average length of cells, length variance, and the ratio of row/column span. Content features include HTML features (based on HTML tags) and lexical features (based on cell content). As a follow-up work, Lautert et al. [2013] additionally consider the category obtained in [Crestan and Pantel, 2011] as one features to further classify tables into a multi-layer taxonomy. The first layer of classification is similar to the one in [Crestan and Pantel, 2011]. A second layer of classification focuses on relational knowledge, by additionally dividing relational knowledge tables into concise, nested, multivalued (simple or composed), and split tables. Lehmberg et al. [2016] construct a web table corpus from Common Crawl (WDC Web Table Corpus, cf. 2.1.3). First, they filter out non-genuine tables (referred to as not innermost tables, i.e., tables that contain other tables in their cells) and tables that contain less than 2 columns or 3 rows. Then, using the table extraction framework of DWTC, the filtered tables are classified as either relational, entity matrix, or layout tables [Eberius et al., 2015]. Recently, deep learning methods have also been used for table type classification. For example, Nishida et al. [2017] regard a table as a matrix of texts, which is similar to an image. Utilizing the type taxonomy from [Crestan and Pantel, 2011], they design a framework named TabNet, consisting of RNN Encoder, CNN Encoder, and Classifier. The RNN Encoder encodes the input table cells to create a 3D table volume, like image data, in the first step. The CNN encoders encode the 3D table volume to capture table semantics, which is used for table type classification by the Classifier. Even though TabNet is designed to capture table structure, it can be applied to any matrix for type classification.

2.2.2 Wikipedia Table Extraction

Wikipedia tables may be regarded as a special case of web tables. They are much more homogeneous than regular web tables and are generally of high quality. Therefore, no additional data cleaning is required. Bhagavatula et al. [2015] construct a Wikipedia table corpus, consisting of 1.6 million tables, with the objective of extracting machine-understandable knowledge from tables. For details, we refer to Sect. 2.1.3.

2.2.3 Spreadsheet Extraction

The Web contains a great variety and number of Microsoft Excel *spreadsheets*. Spreadsheets are often roughly relational. Chen and Cafarella [2013] design an automatic system to extract relational data, in order to support data integration operations, such as joins. A *data frame* is defined as a block of numerical data. Chen and Cafarella [2013] extract 410,554 Microsoft Excel files from the ClueWeb09 Web crawl by targeting Excel-style file endings that contain a data frame. Within a data frame, the attributes might lie on


All-time (List of Grand Slam men's singles champions)

Rank	Player	Total	Years
1	Roger Federer	20	2003–2018
2	Rafael Nadal	17	2005–2018
3	Pete Sampras	14	1990–2002
	Novak Djokovic	14	2008–2018
5	Roy Emerson	12	1961–1967

http://dbpedia.org/page/Rafael_Nadal
B
<Peter_Sampras, careerYears, 1990-2002>
C
Person
A
http://dbpedia.org/ontology/Person

Figure 2.3: Illustration of table interpretation: (A) Column Type Identification. (B) Entity Linking. (C) Relation extraction.

the left or top. Chen and Cafarella [2013] find that 50.5% of the spreadsheets contain a data frame and 32.5% of them have hierarchical top or left attributes (the rest are called *flat* spreadsheets). Among the 49.5% non-data frame spreadsheets, 22% are relational, 10.5% are forms, 3.5% are diagrams, 3% are lists, and 10.5% are other spreadsheets. For each spreadsheet, the extraction system firstly finds the data frame, then extracts the attribute hierarchy (top or left), and finally builds relational tuples (see Sect. 2.3.3 for more details).

2.3 Table Interpretation

Table interpretation encompasses methods that aim to make tabular data processable by machines. Specifically, it focuses on interpreting tables with the help of existing knowledge bases. Bhagavatula et al. [2015] identify three main tasks aimed at uncovering table semantics: (i) *column type identification*, that is, associating a table column with the type of entities or relations it contains, (ii) *entity linking*, which is the task of identifying mentions of entities in cells and linking them to entries in a reference knowledge base, and (iii) *relation extraction*, which is about associating a pair of columns in a table with the relation that holds between their contents. Table 2.5 provides an overview of studies addressing either or all of these tasks.

2.3.1 Column Type Identification

In relational tables, the *core column* (also referred to as *subject column*, *name column*, or *entity column* [Lehmberg and Bizer, 2016]) is a special column that contains entities. Commonly, this is the leftmost column in a table (and other table columns correspond to attributes or relationships of these entities). The identification of the core column is a central pre-processing step for entity linking, table augmentation, and relation extraction. Most of

Table 2.5: Overview of table interpretation tasks addressed in various studies.

Reference	Column type identification	Entity linking	Relation extraction
Bhagavatula et al. [2015]		✓	
Chen and Cafarella [2013]			✓
Efthymiou et al. [2017]		✓	
Fan et al.	✓		
Hassanzadeh et al. [2015]		✓	
Ibrahim et al. [2016]		✓	
Lehmberg and Bizer [2016]	✓		
Limaye et al. [2010]	✓	✓	
Muñoz et al. [2014]			✓
Mulwad et al. [2013]			✓
Mulwad et al. [2010]	✓	✓	✓
Ritze and Bizer [2017]		✓	
Ritze et al. [2016]		✓	
Sekhavat et al. [2014b]			✓
Venetis et al. [2011]	✓		✓
Wang et al. [2012]	✓		
Wu et al. [2016]		✓	
Zhang and Chakrabarti [2013]	✓	✓	
Zhang [2017]	✓	✓	✓
Zwicklbauer et al. [2013]			✓

the existing work assumes the presence of a single core column. Such tables are also known as single-concept relational tables. However, in some cases, a relational table might have multiple core columns that may be located at any position in the table [Braunschweig et al., 2015b], called a multi-concept relational table. Braunschweig et al. [2015b] extend a single-concept method, which utilizes table headings as well as intrinsic data correlations, with more features, like the correlation with the left neighbor, to determine all the core columns. We focus on single-concept relational tables in the remainder of this section.

Generally, *column type identification* is concerned with determining the types of columns, including locating the core column. This knowledge can then be used to help interpret a table. Table 2.6 displays a summary of the methods, which we shall discuss below. Venetis et al. [2011] argue that the meaning of web tables is "only described in the text surrounding them. Header rows exist in few cases, and even when they do, the attribute names are typically useless" [Venetis et al., 2011]. Therefore, they add annotations to tables to describe the sets of entities in the table (i.e., column type identification). This is accomplished by leveraging an IS-A database of entity-class pairs. This IS-A database is created by aggregating all the entity-class $\langle e, C \rangle$

Table 2.6: Comparison of column type identification tasks.

Reference	Knowledge base	Method
Venetis et al. [2011]	Automatically built IS-A database	Majority vote
Mulwad et al. [2010]	Wikitology	Entity search
Fan et al.	Freebase	Concept-based method + crowdsourcing
Wang et al. [2012]	Probase	Heading-based search
Lehmberg and Bizer [2016]	DBpedia	Feature-based classification
Zhang [2017]	Wikipedia	Unsupervised featured-based method
Zhang and Chakrabarti [2013]	-	Semantic graph method

pairs that are mined from the Web (100 million English documents using 50 million anonymized queries) using the pattern "*C* [such as—including] *e* [and—,—.]." A class label is assigned to a column if a certain fraction of entities in that column is identified with that label in the IS-A database. Venetis et al. [2011] conclude that using a knowledge base (YAGO) results in higher precision, while annotating against the IS-A database has better coverage, i.e., higher recall. Mulwad et al. [2010] map each cell's value in a column to a ranked list of classes, and then selects a single class which best describes the whole column. To get the ranked list of classes, a complex query, based on cell values, is submitted to the Wikitology knowledge base [Syed, 2010]. Possible class labels are obtained by utilizing the relevant entities in the knowledge base. Then, a PageRank-based method is used to compute a score for the entities' classes, from which the one with the highest score is regarded as the class label. Mapping each column to one of the four types ("Person", "Place", "Organization," and "Other"), Mulwad et al. [2010] achieve great success on "Person" and "Places," and moderate success on "Organization" and "Other" types, due to their sparseness in the reference knowledge base.

Because of the inherent semantic heterogeneity in web tables, not all tables can be matched to a knowledge base using pure machine learning methods. Fan et al. propose a "two-pronged" approach for matching web tables' columns to a knowledge base. First, a concept-based method is used to map each column to the best knowledge base concept. Specifically, they employ Freebase as the concept catalog. Second, a hybrid human-machine framework discerns the concepts for some exceptional columns manually. The matches between table columns and their candidate concepts are represented as a bipartite graph, where relationships correspond to edges. Fan et al. employ crowdsourcing for this task, and find that a higher payment leads to better accuracy.

A table corpus is constructed in [Wang et al., 2012] and it is classified according to a probabilistic taxonomy called Probase, which is able to understand entities, attributes, and cells in tables. To get the table semantics, a top-*k* candidates concepts are returned based on the table headings, which is similar to the idea in [Limaye et al., 2010] (cf. Sect. 2.3.2). The candidate concepts assist to detect entities in a given column by computing the maximum number of common concepts. In turn, the entity column type is obtained based on the confidence of the concepts. Wang et al. [2012] demonstrate that table headers can help to understand the columns as well as to identify the core column.

Lehmberg and Bizer [2016] propose a categorization scheme for web table columns which distinguishes the different types of relations that appear in tables on the Web. First, a binary relation is a relation that holds between the core column and the values in another column, e.g., populations of cities. Second, an N-ary relation is a relation that holds between the core column and additional entities and values in other columns. Third, an independent column is one that has no direct relation with the core column. Lehmberg and Bizer [2016] propose a feature-based classifier that distinguishes between these three types of relations for better table interpretation.

Zhang [2017] presents TableMiner+ for semantic table interpretation, where core column detection and type identification linking are executed at the same stage. Zhang [2017] first simply uses regular expressions and classifies cells as "empty," "entities," "numbers," "data," "text," or "other." Then, evidence is gathered from the Web for each column to predict the likelihood of it being subject. Specifically, a keyword query is composed from all text content in each row, and the subject entity in this row is detected by recognizing the top-ranked page. Finally, an unsupervised feature-based method is employed to find the core column and type by aggregating evidence across all rows. Features include the fraction of empty cells, the fraction of cells with unique content, context match score (heading frequency within surrounding text), and web search score. The main differences between TableMiner+ and other methods are twofold: (1) TableMiner+ uses context outside the tables while others not, and (2) it adopts an iterative process to optimize and enforce the interdependence between different annotation tasks (entity linking and relation extraction.)

The above methods work well for string values and static attributes but perform poorly for numeric and time-varying attributes. Zhang and Chakrabarti [2013] build a semantic graph over web tables suited for numeric and time-varying attributes by annotating columns with semantic labels, like timestamp, and converting columns by comparing with columns from other tables. While this method is designed for entity augmentation, it can also be

Table 2.7: Comparison of entity linking tasks.

Reference	Knowledge base	Method
Limaye et al. [2010]	YAGO catalog, DBpedia, and Wikipedia tables	Inference of five types of features[a]
Bhagavatula et al. [2015]	YAGO	Graphical model
Wu et al. [2016]	Chinese Wikipedia, Baidu Baike, and Hudong Baike	Probabilistic method[b]
Efthymiou et al. [2017]	DBpedia	Vectorial representation and ontology matching
Zhang [2017]	Wikipedia	Optimization
Mulwad et al. [2010]	Wikitology	SVM classifier
Lehmberg et al. [2016]	Google Knowledge Graph	-
Ibrahim et al. [2016]	YAGO	Probabilistic graphical model
Zhang et al. [2013]	DBpedia	Instance-based schema mapping
Hassanzadeh et al. [2015]	DBpedia, Schema.org, YAGO, Wikidata, and Freebase	Ontology overlap[c]
Ritze and Bizer [2017]	DBpedia	Feature-based method
Ritze et al. [2015, 2016]	DBpedia	Feature-based method
Lehmberg and Bizer [2017]	DBpedia	Feature-based method

[a] Designed for table search
[b] Multiple KBs
[c] KB comparison

utilized for column type identification.

2.3.2 Entity Linking

Recognizing and disambiguating specific entities (such as persons, organizations, locations, etc.), a task commonly referred to as *entity linking*, is a key step to uncovering semantics [Bhagavatula et al., 2015]. Since many web tables are relational, describing entities, entity linking is a key step to understanding what the table is about. A number of table-related tasks, such as table population [Wang et al., 2015a], and table search, rely on entity linking in tables. Table 2.7 compares the tasks we will discuss below.

Limaye et al. [2010] pioneered research on table entity linking. They introduce and combine five features, namely, the TF-IDF scores between cell text and entity label, the TF-IDF scores between the column header and the type label, the compatibility between column type and cell entity, compatibility between relation and pair of column types, and the compatibility between relation and entity pairs. Their idea of factor graph-based entity linking approach influenced later research. For example, Bhagavatula et al. [2015] design a system called TabEL for table entity linking. TabEL employs a graphical model that "assigns higher likelihood to sets of entities that tend to co-occur in Wikipedia documents and tables" [Bhagavatula et al., 2015]. Specifically, it uses a supervised learning approach and annotated mentions in tables for training. TabEL focuses on Wikipedia table and executes mention identification for each table cell, then obtains a set of candidate entities for disambiguation. The disambiguation technique is based on the assumption that entities in a given row and column tend to be related. They use a collective classification technique to optimize a global coherence score for a set of entities in a given table. By comparing against traditional entity linking methods for unstructured text, Bhagavatula et al. [2015] demonstrate the need for entity linking methods designed specifically for tables.

Unlike most methods, which consider a single knowledge base, Wu et al. [2016] propose an entity linking method for web tables that considers multiple knowledge bases to ensure good coverage. From each knowledge base, entities whose names share at least one word with the content of a given table cell are taken as candidates. Then, an entity disambiguation graph is constructed, consisting of mention nodes, entity nodes, mention-entity edges, and entity-entity edges. The method utilizes entity linking "impact factors," which are two probabilities, for ranking candidates and for disambiguating entities, based on mention nodes and edges. To incorporate multiple knowledge bases, "same-As" relations between entities from different knowledge bases are leveraged to reduce errors and to improve coverage. This system shares many similarities with TabEL. TabEL, however, does not consider synonyms and deals with a single KB. Efthymiou et al. [2017] propose three unsupervised annotation methods for matching web tables with entities. The first is a lookup-based method, which relies on the minimal entity context from the tables to discover correspondences to the knowledge base. A second method exploits a vectorial representation of the rich entity context in a knowledge base to identity the most relevant subset of entities in web tables. The third method is based on ontology matching, and exploits schematic and instance information of entities available both in a knowledge base and in a web table. Efthymiou et al. [2017] find that the hybrid methods that combines the second and third methods (in any order) tend to perform best. The column type identification component of TableMiner+ [Zhang, 2017] has already been discussed earlier, in Sect. 2.3.1. Building on this, TableMiner+ uses the partial annotations from column type identification for all columns to guide entity linking in the rest of the table. It re-ranks table rows under the assumption that some cells are easy to disambiguate, i.e., they have more candidates or the text is less ambiguous (candidate sampling). In each iteration of this so-called *learning* phase, it searches new candidates and compares the feature representation of each candidate entity (web search results) against all the feature representations of that cell (using the same features as for column type identification). The associated concepts with the highest scoring entity are gathered as candidate concepts for the column. These are further compared against those from the previous iteration in the *learning* phase (optimization). The process is repeated until convergence is reached.

Mulwad et al. [2010] exploit the predicted class labels for columns (see Sect. 2.3.1) as additional evidence, to link entities in table cells. A knowledge base is queried to construct a feature vector, which comprises the entity's retrieval score, Wikipedia page length, PageRank, etc., which are used for computing the similarity score against the table cell's value. The feature vectors are input to an SVMRank classifier, which outputs a ranked list of entities. The top-ranked entity is selected and is used to introduce two more features for a final classification (the SVM rank score for the top-ranked entity and the score difference between the top two entities). The final classification yields a binary outcome whether the entity should be linked or not. Similar to the column type identification task, this method performs very well on the "Person" and "Place" entity types, achieves moderate accuracy on "Organization," and low accuracy on "Other" (for the same reason of sparseness, as before). A similar approach is taken by Lehmberg et al. [2016], but they perform entity linking in table cells first, using the Google Knowledge Graph, and then use this information for getting class labels for columns.

Another study on knowledge base matching in [Ibrahim et al., 2016] aims to overcome the problem of table matching and aggregation by making sense of entities and quantities in web tables. Ibrahim et al. [2016] map the table elements of table headers, entity tables cells and numeric table cells to different knowledge bases. Specifically, (i) tables headers denote classes or concepts and are linked to a taxonomic catalog or to Wikipedia pages, (ii) named entities are mapped to a knowledge base (YAGO), and (iii) numeric cells, which denote quantities, are mapped to normalized representations. An interesting observation made about quantity linking is that many of the linking errors are (i) due to the absence of specific measures or units and (ii) because of ambiguous headings, like "Nat."

As mentioned in Sect. 2.2.1, a *relational table* refers to an entity-attribute table, where a set of entities and their attributes are listed. Zhang et al. [2013] propose an instance-based schema mapping method to map entity-attribute tables to a knowledge base. In [Zhang et al., 2013], an entity-attribute table is supposed to have a key column, which contains a set of entities. Each tuple is an entity with its attributes. Then, memory-based indexes are used to judge whether a tuple contains candidate entities, resulting in an evidence mapping vector. This vector is then used for finding the table-to-KB schema mapping, which essentially serves as a bridge between web tables and knowledge bases.

The choice of the knowledge base for uncovering table semantics is important. Hassanzadeh et al. [2015] give a detailed study on the utility of different knowledge bases, including DBpedia, Schema.org, YAGO, Wikidata, and Freebase. The method of concept linking in [Hassanzadeh et al., 2015] is tagging columns with entity types (classes) in the knowledge base. Specifically, they firstly get the basic statistical distribution of tables sizes and values. Then, with the help of the selected knowledge base, the distribution of overlap scores in the ontology is obtained. Finally, these scores can give an indication of how well the table's content is covered by the given knowledge base.

Ritze and Bizer [2017] study the utility of different features for matching tables to DBpedia. These features are extracted from the table itself (such as entity label, attribute label, value, entity, set of attribute labels, table, URL, page title, and surrounding text) or from the knowledge base (such as instance label, property label, class label, value, instance count, instance abstract, and classes). The problem is decomposed into three specific subtasks: table-to-class matching, row-to-instance matching, and attribute-to-property matching. Ritze and Bizer [2017] introduce a specific similarity matcher for each feature, resulting in similarity matrices, representing the feature-specific results. These matrix predictors can be used to decide which features to use for which web table. Ritze et al. [2016] implement the T2K Match framework [Ritze et al., 2015] to map the WDC Web corpus to DBpedia, for knowledge base extension. Taking table content as evidence, the incomplete and unclear values of DBpedia can be filled and corrected. They find that "only 1.3% of all tables that were extracted from the Web crawl contained relational data. Out of these relational tables, about 3% could be matched to DBpedia" [Ritze et al., 2016]. Ritze et al. [2016] further introduce three fusion strategies for deciding which value to use as output. Their best method achieves an F1-score of 0.7. However, the above methods tend to perform better for large tables, i.e., tables with several rows. It is considered as one of the main limitations of methods matching tables to DBpedia. To overcome this, Lehmberg and Bizer [2017] stitch tables, i.e., merge tables from the same website as a single large table, in order to improve the matching quality.

2.3.3 Relation Extraction

Relation extraction refers to the task of associating a pair of columns in a table with the relation that holds between their contents and/or extracting relationship information from tabular data and representing them in a new format (e.g., RDF). Table 2.8 summarizes the methods we will discuss below.

Table 2.8: Comparison of relation extraction tasks.

Reference	**Knowledge base**	**Method**	**Source of extraction**
Venetis et al. [2011]	IS-A database	Frequency-based	Core + attribute columns
Mulwad et al. [2010]	DBpedia	Utilizing CTI and EL	Any pair of columns
Mulwad et al. [2013]	DBpedia	Semantic passing	Any pair of columns
TableMiner+ [Zhang, 2017]	Wikipedia	Optimization	Any pair of columns
Sekhavat et al. [2014b]	YAGO, PATTY		Any pair of entities in the same row
Muñoz et al. [2014]	DBpedia	Look-up based	Any pair of entities in the same row
Zwicklbauer et al. [2013]	DBpedia	Frequency-based	
Chen and Cafarella [2013]	-	Classification	All columns

Venetis et al. [2011] add annotations to tables to describe the binary relationships represented by columns. This is accomplished by leveraging a relations database of (argument1, predicate, argument2) triples. For binary relationships, the relationship between columns A and B is labeled with R if a substantial number of pairs of values from A and B occur in the relations database. Venetis et al. [2011] are only able to annotate a small portion of a whole table corpus (i.e., low recall). They discover that the vast majority of these tables are either not useful for answering entity-attribute queries, or can be labeled using a handful of domain-specific methods.

Mulwad et al. [2010] propose a preliminary method for relation extraction, which utilizes the results of entity linking and column type prediction. Specifically, the method generates a set of candidate relations by querying DBpedia using SPARQL. Each pair of strings in two columns vote for the candidate relation. The normalized scores are used for ranking candidate relations and the highest one is taken as the column relation. In follow-up work, Mulwad et al. [2013] implement an improved semantic message passing method to extract RDF triples from tables. The semantic message passing first pre-processes the input table, separated by table elements such as column headers, cell values, columns, etc. Then, the processed table is passed to a *query and rank* module, which turns to knowledge bases from Linked Open Data to find candidates for each table element. Finally, a *joint inference* step uses a probabilistic graph model to rank candidate relations that were identified for the table elements. Mulwad et al. [2013] point out that current methods rely on semantically poor and noisy knowledge bases and can only interpret part of a table (low recall). Moreover, systems for numeric values remain challenging, which is consistent with [Ibrahim et al., 2016].

TableMiner+ [Zhang, 2017] interprets relations between the core column and other columns on each row independently. It computes an individual confidence score for each candidate relation from each row. The candidate set of relations for two columns is derived by collecting the winning relations on all rows. A final confidence score of a candidate relation adds up its instance and context score computed based on context overlap. It is used to find the relation with the highest confidence. A key finding in [Zhang, 2017] is that a system that is based on partial tabular data can be as good as systems that use the entire table.

Relation extraction can also be used to augment Linked Data repositories [Sekhavat et al., 2014b]. Sekhavat et al. [2014b] propose a probabilistic approach using under-explored tabular data. Assuming that the entities co-occurring in the same table are related, they focus on extracting relations between pairs of entities appearing in the same row of a table. Entities in table cells are mapped to a knowledge base first. Then, sentences containing both entities from the same table row are collected from a text corpus. Next, textual patterns (describing these two entities' relations) are extracted. Finally, the probability of the possible relations is estimated using Bayesian inference. A new relation, which is a triple consisting of two entities and a pattern, can be added to the Linked Data repository for augmentation. Muñoz et al. [2014] utilizes entity annotations in Wikipedia tables. Taking existing relations between entities in DBpedia, they look these entities up in Wikipedia tables. This then indicates that the same relation stands between entities in other rows of this table.

Zwicklbauer et al. [2013] propose a simple method to annotate table headings with semantic types, using DBpedia's type system. The method is divided into three steps: (i) table cell entity linking, using a search-based disambiguation method (detailed in Sect. 2.3.2), (ii) entity type resolution (looking up the corresponding entity types from DBpedia), and (iii) type aggregation, which takes the union of all entity types in that column and selects the most frequent of those as the type for the given table heading. Despite being considerably simpler, the method in [Zwicklbauer et al., 2013] achieves comparable accuracy to other methods, such as Venetis et al. [2011]. The authors attribute it DBpedia being more exhaustive and containing high quality data.

Chen and Cafarella [2013] introduce a system to automatically extract relational data from spreadsheets instead of the Web. Most of the methods on spreadsheets requires users to provide sheet-specific specific rules [Hung et al., 2011]. In contrast, Chen and Cafarella [2013] realize it in an automatic manner. Generally, the system detects attributes and values, identifies the hierarchical structure of attributes, and generates relational tuples from spreadsheet data. Specifically, the so-called *frame finder* module of their system aims to identify the data frame regions within a spreadsheet. These data frames consist of attribute and value regions. First, it labels each row with one of the categories: title, header, data, or footnote. Then, the data frame regions are created, which are passed to the *hierarchy extractor* for recovering the attribute hierarchies by finding all parent-child pairs in an attribute region. Finally, a series of parent-child candidates are generated and the true parent-child pairs are identified through classification. Alternatively, a so-called enforced-tree classification is proposed, which constructs a strict hierarchical tree for attributes. In the end, relational tuples are generated from the value region, whose value is annotated with one attribute from the attribute hierarchy.

2.3.4 Other Tasks

Data translation is concerned with the problem of mapping raw data, collected from heterogenous sources, to a transformed version for the end user [He et al., 2018]. Tables encode a large number of mapping relationships as column pairs, e.g., person and birthday, which can be useful data assets for data translation. Wang and He [2017] propose to automatically synthesize mapping relationships using table corpora by leveraging the compatibility of tables based on co-occurrence statistics. Braunschweig et al. [2015b] propose a method to normalize web tables in cases where multiple core columns and mixed concepts are detected in one table.

Web tables are embedded in HTML pages, where the surrounding text can help to understand what a given table is about. However, these surrounding sentences are not equally beneficial for table understanding. Wang et al. [2015b] present the Table-Related Context Retrieval system (TRCR) to determine the relevance between a table and each surrounding sentence. Using TRCR, the most relevant texts are selected to uncover table semantics. Another related study is performed in [Govindaraju et al., 2013], where NLP tools, like part-of-speech tagging, dependency paths, and named-entity recognition, are explored to mine surrounding texts for understanding table semantics. Braunschweig et al. [2015a] propose a heuristic approach that extracts text snippets from the context of a web table, i.e., caption, headline, surrounding text, and full text, which describe individual columns in the table and link these new labels to columns. As a follow-up, Braunschweig et al. [2016] propose a contextualization method of splitting table context into paragraphs with consistent topics, providing a similarity measure that is able to match each paragraph to the table in question. Paragraphs are then ranked based on their relevance.

2.4 Table Search

Table search is the task of returning a ranked list of tables in response to a query. It is an important task on its own and is regarded as a fundamental step in many other table mining and extraction tasks as well, like table integration or data completion. Table search functionality is also available in commercial products; e.g., Microsoft Power Query provides smart assistance features based on table search. Depending on the type of the query, table search may be classified as *keyword query search* and *table query search*.

2.4.1 Keyword Query Search

Given a keyword query, the process of returning a ranked list of tables is called *keyword query search*. One of the first published methods is by Cafarella et al. [2008a], who implement keyword table search on top of an existing web search engine. Specifically, they extract the top-*k* tables from the returned web pages. In follow-up work, a similar system called OCTOPUS [Cafarella et al., 2009] extends the same method (referred to as SimpleRank) with a reranking mechanism (SCPRank) that considers attribute co-occurrences.

Later works search directly within a designated table corpus. Methods may be divided into *document-based* and *feature-based* approaches. According to the first group of approaches, a document-based representation is created for each table. This might contain all text included in the table or only certain elements of the table (e.g., caption or header labels). Then, these document-based representations may be ranked using traditional retrieval models, such as TF-IDF [Pimplikar and Sarawagi, 2012].

Feature-based methods employ supervised machine learning for table ranking. Features may be divided into three main categories: query features, table features and query-table features. *Query features* include query length and IDF scores of query terms. *Table features* characterize the table in terms of its dimensions (number of rows, columns) and schema coherency. With a focus on Wikipedia tables, Bhagavatula et al. [2013] introduce features related to the connectivity of the Wikipedia page (pageViews, inLinks, and outLinks) and the table's importance within the page (table importance and table page fraction). Finally, *query-table features* capture the degree of matching between the user's information need and the table. Typically, these include similarity scores between the query and various table elements. Table 2.9 lists a selection of features for keyword table search. In terms of learning algorithm, Cafarella et al. [2008a] train a linear regression classifier, while Bhagavatula et al. [2013] train a linear ranking model learned with Coordinate Ascent.

Instead of relying on a single keyword query as input, Pimplikar and Sarawagi

Table 2.9: A selection of features for keyword table search.

Query features		**Source**
QLEN	Number of query terms	[Tyree et al., 2011]
IDF$_f$	Sum of query IDF scores in field f	[Qin et al., 2010]
Table features		
#rows	Number of rows in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
#cols	Number of columns in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
#of NULLs in table	Number of empty table cells	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
PMI	ACSDb-based schema coherency score	[Cafarella et al., 2008a]
inLinks	Number of in-links to the page embedding the table	[Bhagavatula et al., 2013]
outLinks	Number of out-links from the page embedding the table	[Bhagavatula et al., 2013]
pageViews	Number of page views	[Bhagavatula et al., 2013]
tableImportance	Inverse of number of tables on the page	[Bhagavatula et al., 2013]
tablePageFraction	Ratio of table size to page size	[Bhagavatula et al., 2013]
Query-table features		
#hitsLC	Total query term frequency in the leftmost column cells	[Cafarella et al., 2008a]
#hitsSLC	Total query term frequency in second-to-leftmost column cells	[Cafarella et al., 2008a]
#hitsB	Total query term frequency in the table body	[Cafarella et al., 2008a]
qInPgTitle	Ratio of the number of query tokens found in page title to total number of tokens	[Bhagavatula et al., 2013]
qInTableTitle	Ratio of the number of query tokens found in table title to total number of tokens	[Bhagavatula et al., 2013]
yRank	Rank of the table's Wikipedia page in web search engine results for the query	[Bhagavatula et al., 2013]
MLM similarity	Language modeling score between query and multi-field document repr. of the table	[Hasibi et al., 2017a]

[2012] take q columns, each described by a set of keywords $Q_1, \ldots, Q_q$, as input (e.g., $Q_1 =$"chemical element," $Q_2 =$"atomic number," and $Q_3 =$"atomic weight"), and return a table with q columns as the answer. First, they rank tables using the union of words in $Q_1, \ldots, Q_q$. Then, each table column is labeled with the query column it maps to. Finally, relevant columns and rows are merged into a single table, by considering the table-level relevance scores and the column-level mapping confidence scores. To decide if two rows are duplicates of each other, they employ the method in [Gupta and Sarawagi, 2009]. In Chapter 3 we will perform semantic matching between queries and tables for keyword table search. Specifically, we (i) represent queries and tables in multiple semantic spaces (both discrete sparse and continuous dense vector representations) and (ii) introduce various similarity measures for matching those semantic representations. Most recently, Deng et al. [2019] train word embeddings utilizing the Wikipedia table corpus and achieve comparable results.

2.4.2 Search by Table

Table search is not limited to keyword queries. The input may be also be a table, in which case the task of returning related tables is referred to as *search by table* or *query by table*. At its core, this task boils down to computing a similarity score between the input and candidate tables, which we shall refer to as *table matching*. Search by table may be performed for different goals: (1) to be presented to the user to answer her information need [Das Sarma et al., 2012, Limaye et al., 2010, Nguyen et al., 2015] and (2) to serve as an intermediate step that feeds into other tasks, like table augmentation [Ahmadov et al., 2015b, Lehmberg et al., 2015, Yakout et al., 2012, Nargesian et al., 2018].

One group of approaches addresses the table matching task by using certain table elements as a keyword query, and scoring tables using keyword-based methods. For example, Ahmadov et al. [2015b] use table entities and table headings as queries to retrieve a ranked list of tables. The two ranked lists are then intersected afterwards in order to arrive at a more complete result set.

More commonly, table matching is tackled by dividing tables into various elements (such as table caption, table entities, column headings, cell values), then computing element-level similarities. Table 2.10 provides an overview of the table elements that have been utilized in past work. It is worth pointing out that in most of these cases, table search is not the ultimate goal, it is only used as a component in a larger application. The Mannheim Search Join Engine [Lehmberg et al., 2015] seeks to extend the input table with additional attributes. It utilizes table headings by comparing the column headings between the input table and candidate tables. Specifically, they first

Table 2.10: Overview of table elements used when querying by table for various table-related applications.

Application	**Source**	T_E	T_H	$T_{[:,j]}$	T_p	$T_{[i,j]}$
Data completion	[Ahmadov et al., 2015b]	√	√			
Relation join	[Lehmberg et al., 2015]		√			
Schema complement	[Das Sarma et al., 2012]	√	√			
Entity complement	[Das Sarma et al., 2012]	√				
Table augmentation	[Yakout et al., 2012]		√	√	√	√
Diverse table search	[Nguyen et al., 2015]		√			√
Table cell retrieval	[Limaye et al., 2010]			√		√

filter tables that share at least one column heading with the input table, using exact term matching. Then, the table matching score is computed by (i) building an edit distance similarity matrix between the input and candidate tables' column headings, and (ii) calculating the Jaccard similarity of the two tables using the matrix's maximum weighted bipartite matching score. Similar to the Mannheim Search Join Engine that is based on table headings, Nargesian et al. [2018] search tables that are likely unifiable with the seed table, which is called *attribute union ability*. Nargesian et al. [2018] formalize three statistical models to estimate the likelihood that two attributes contain values that are in the same domain. The simplest case, named *set domains*, uses the size of the intersection of values between two columns. The second case, called *semantic domains*, measures the semantic similarity between the values by mapping the columns to classes, e.g., entities. For values that are expressed in natural language, the third case of *natural language domains* measures semantics based on natural langue rather than on ontologies. They use word embeddings trained based on Wikipedia documents to define natural language domains and statistical tests between the vectors are used to evaluate the likelihood that two attributes are from the same domain. Das Sarma et al. [2012] aim to find related tables for augmenting the input table with additional rows or columns, referred to as *entity complement* and *schema complement*, respectively. Entity complement considers the relatedness between entity sets of the input and candidate tables. Relatedness between two entities is estimated by representing entities as weighed label sets (from a knowledge base or from a table corpus) and taking their dot product. Das Sarma et al. [2012] propose multiple methods to aggregate pairwise entity relatedness scores for computing relatedness between two sets of entities. Schema complement combines two element-wise similarities: table entities and column headings. The former considers the overlap between table entities. The latter estimates the benefits of adding a column from the candidate table to the input table by determining the consistency between the new column and the existing columns of the input table. Yakout et al. [2012] propose InfoGather, a holistic method for matching tables in order to support three core operations: augmentation by column headings, augmentation by example, and column heading discovery. They consider element-wise similarities, including table context, URL, tuples, column headings, column values, and table data, as well as cross-element similarity between table and context. Similarity is measured using the vector product of TF/IDF-weighted term vectors. Then, element-level similarity scores are combined as features in a machine learned model. In follow-up work, InfoGather is extended as InfoGather+ [Zhang and Chakrabarti, 2013] to incorporate tables with numeric and time-varying attributes. Nguyen et al. [2015] consider the diversity of the returned tables. They focus on two table elements: column headings and table data. The former is similar in spirit to the Mannheim Search Join Engine [Lehmberg et al., 2015]. The latter works by measuring the similarity between table columns, which are represented as term frequency vectors. In Chapter 4, we will perform semantic table retrieval when the query is a table.

Unlike the above methods, which consider tables as the unit of retrieval, Limaye et al. [2010] return a ranked list of cells as result. They train a machine learning method for annotating (i) entities in tables cells, (ii) columns with types, and (iii) relations between columns. Then, search is performed by issuing an automatically generated structured query.

2.5 Question Answering on Tables

Tables are a rich source of knowledge that can be utilized for answering natural language questions. This problem has been investigated in two main flavors: (i) where the table, which contains the answer to the input question, is given beforehand [Pasupat and Liang, 2015c], and (ii) where a collection of tables are to be considered [Sun et al., 2016]. The latter variant shares many similarities with traditional question answering (QA), while the former requires different techniques. One of the main challenges of QA on tables, shared by both scenarios, is how to match the unstructured query with the (semi-)structured information in tables. Question answering on tables is also closely related to work on natural language interfaces to databases, where the idea is that users can issue natural language queries, instead of using formal structured query languages (like SQL), for accessing databases [Androutsopoulos et al., 1995, Li and Jagadish, 2014, Li et al., 2005, Popescu et al., 2003]. *Semantic parsing* is the task of parsing natural language queries into a formal representation. Semantic parsing is often used in question answering, by generating logical expressions that are executable on knowledge bases [Berant et al., 2013, Fader et al., 2014].

2.5.1 Using a Single Table

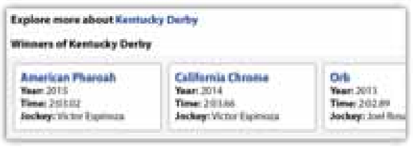
We first discuss approaches that take a single table as input (sometimes referred to as *knowledge base table* [Yin et al., 2016]), and seek the answer to the input question in that table. The basic idea is to regard the input table as a knowledge base, which poses a number of challenges. First, knowledge bases contain a canonicalized set of relations, while tabular data is much more noisy. Second, traditional semantic parsing sequentially parses natural language queries into logical forms and executes them against a knowledge base. To make them executable on tables, special logical forms are required. Lastly, semantic parsing and query execution become complicated for complex questions as they need carefully designed rules to parse them into logic forms. Pasupat and Liang [2015a] propose to answer complex questions, involving operation such as comparison, superlatives, aggregation, and arithmetics in order to address above problems. They convert the input table into a knowledge graph by taking table rows as row nodes, strings as entity nodes, and columns as directed edges. The column headings are used as predicates. Numbers and strings are normalized following a set of manual rules. Being one of the earliest works addressing this task, Pasupat and Liang [2015a] follow a traditional parser design strategy. A semantic parser is trained based on a set of question-answer pairs. Given a table and a question, a set of candidate logical forms is generated by parsing the question. Then, logic forms are ranked using a feature-based representation, and the highest ranked one is applied on the knowledge graph to obtain the answer. Pasupat and Liang [2015a] develop a dataset, called WikiTable-Question, which consists of a random sample of 2,100 tables from Wikipedia and 22,000 question-answer pairs. The proposed approach is found to suffer from the coverage issue, i.e., it is able to answer only 20% of the queries that have answers in Freebase.

Different from semantic parsing methods that require predefined logical operations, Yin et al. [2016] propose a neural network architecture, named Neural Enquirer for semantic parsing with a specific table. Neural Enquirer is a fully neural system that generates distributed representations of queries and tables (called query encoder and table encoder, respectively). Then, the question is executed through a series of query operations, called executors, with intermediate execution results computed in the form of table annotations at different levels. Training can be performed in an end-to-end fashion or carried out using step-by-step supervision (for complex queries). They use query answers as indirect supervision, but jointly perform semantic parsing and query execution in distributional spaces. The distributed representations of logical forms are learned in end-to-end tasks, which is based on the idea of adopting the results of the query execution as indirect supervision to train the parser. It is worth pointing out that this model makes a number of strong assumptions. For example, they consider four specific types of queries, provide the logical form template for each, and carefully and manually select a table that is supplied as part of the input. Similar to [Yin et al., 2016], Neelakantan et al. [2015] attempt to solve the task of question answering on tables using neural networks, a system called Neural Programmer. Neural Programmer runs for T steps and the final result is formed step by step. The model adopts a Recurrent Neural Network (RNN) architecture to process the input question, a selector to assign probabilities to a set of possible operations and data segments at each step, and a history RNN to remember the previous operation and data selections. Providing a small set of basic operations, they also take the result of correct executions as indirect supervision. During the training, adding random noise to the gradient greatly improves performance as the operations and data selections are quite heterogenous.

2.5.2 Using a Collection of Tables

Another line of work focuses on answering questions using a collection of tables. These approaches are more similar to traditional question answering on text, comprising of candidate identification, query type prediction, and ranking components. The main differences are twofold. One is schema matching, which is the same as before (in Sect. 2.5.1), but there is an additional normalization issue across tables here. The other is the need for extracting quantity values from tables. Sarawagi and Chakrabarti [2014] show that over 40% of table columns contain numeric quantities, and propose a collective extraction framework to extract quantities from raw web tables based on a consensus model. Their system, called QEWT, extends the work of Banerjee et al. [2009] to tables. They employ keyword-based table ranking in order to fetch table candidates. This corresponds to the candidate snippet/answer identification step in traditional QA. QEWT can answer quantity-target queries with a ranked list of quantity distributions, which are taken from the tables. It uses a table column unit annotator based on probabilistic context free grammars for easily extracting quantities from table columns to deal with ambiguity (both for headings and for values). From an information retrieval perspective, quantity queries on web tables refer to the task of returning a ranked list of quantities for a query. QEWT employs a quantity response model for this task.

Inspired by classical textual QA, Sun et al. [2016] decompose table cells into relational chains, where each relational chain is a two-node graph connecting two entities. Specifically, each row of a table represents relations among the cells. They construct a pseudo-node to connect all the cells and take



Explore more about Kentucky Derby

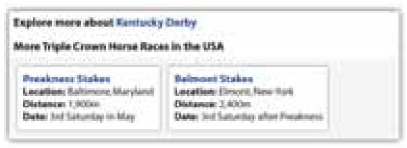
Winners of Kentucky Derby

American Pharoah
Year: 2015
Time: 2:03.02
Jockey: Victor Espinoza

California Chrome
Year: 2014
Time: 2:03.66
Jockey: Victor Espinoza

Orb
Year: 2013
Time: 2:02.89
Jockey: Joel Rosa

(a)



Explore more about Kentucky Derby

More Triple Crown Horse Races in the USA

Preakness Stakes
Location: Baltimore, Maryland
Distance: 1,900m
Date: 3rd Saturday in May

Belmont Stakes
Location: Elmont, New York
Distance: 2,400m
Date: 3rd Saturday after Preakness

(b)

Figure 2.4: Illustration in Chirigati et al. [2016], showing an example of knowledge exploration for the query of "kentucky derby" through Knowledge Carousels: (a) a downward showing the winners of Kentucky Derby; (b) a sideway representing the famous Triple Crown horse races in the US, of which Kentucky Derby is a member.

the headings to label relationships. Any pair of cells in the same row form a directional relational chain. The input query is also represented as a two-node graph question chain, by identifying the entities using an entity linking method. The task then boils down to finding the relational chains that best match the question chain. This matching is performed using deep neural networks, to overcome the vocabulary gap limitation of bag-of-words models. Specifically, they employ the Convolutional Deep Structured Semantic Model (C-DSSM) [Shen et al., 2014]. They find that combining the deep features with some shallow features, like term-level similarity between query and table chains, achieve the best performance. Sun et al. [2016] conclude that their method can complement KB-based QA methods by improving the coverage.

2.6 Knowledge Base Augmentation

The knowledge extracted from tabular data can be used for enriching knowledge bases. First, we present an approach that is devised for exploring the knowledge contained in web tables. Then, we discuss methods for knowledge base augmentation using tabular data.

2.6.1 Tables for Knowledge Exploration

The knowledge contained in web tables can be harnessed for knowledge exploration. Knowledge Carousels [Chirigati et al., 2016] is the first system addressing this, by providing support for exploring "is-A" and "has-A" relationships. These correspond to two kinds of entity-seeking queries (queries searching for attributes and relationships of entities), called "sideways" and "downwards," respectively. Given an input entity, Chirigati et al. [2016] utilize web tables to select the carousel type, create a set entities of this carousel, generate human-readable titles, and rank carousels based on popularity and relatedness extracted from tables. See Fig. 2.4 for an illustration.

2.6.2 Knowledge Base Augmentation and Construction

Tabular data on the Web can be used to construct new or augment existing knowledge bases.

Knowledge Base Augmentation

In Sect. 2.3, we have presented techniques for interpreting tables with the help of knowledge bases. The obtained annotations, in turn, can contribute to extending those knowledge bases. *Knowledge base augmentation*, also known as *knowledge base extension*, is concerned with generating new instances of relations using tabular data and updating knowledge bases with the extracted information.

Knowledge bases need to be complete, correct, and up-to-date. A precondition of extending knowledge bases using web tables is matching them to those existing knowledge bases. Specifically, matching problems include *table-to-class matching*, *row-to-instance matching*, and *attribute-to-property matching*. Ritze et al. [2015] propose an iterative matching method, T2K, to match web tables to DBpedia for augmenting knowledge bases. They also develop and make publicly available the T2D dataset for matching, consisting of 8,700 schema-level and 26,100 entity-level correspondences between web tables and DBpedia, which are extracted and annotated manually. The T2K method utilizes the T2D dataset to execute iterative steps between candidate matching and property matching, to find proper entities/schemas in DBpedia for table rows/columns. However, T2D mainly focuses on large tables and does not work that well for small-sized tables [Lehmberg and Bizer, 2017]. To counter this problem, Lehmberg and Bizer [2017] propose to combine tables from each website into larger tables for table matching, building on the intuition that tables from the same website are created in a similar fashion.

Strictly speaking, we classify the work in Wang et al. [2015a] as *row extension*. Nevertheless, since they map table entities to a knowledge base with the purpose of collecting more entities from other tables that belong to the same concept in the knowledge base, their work can also be regarded as a knowledge base augmentation task.

Knowledge Base Construction

Instead of augmenting existing knowledge bases, web tables contain abundant information to be turned to knowledge bases themselves.

Even though there exists a number of large-scale knowledge bases, they are still far from complete [Dong et al., 2014]. Therefore, Dong et al. [2014] introduce a web-scale probabilistic knowledge base named Knowledge Vault

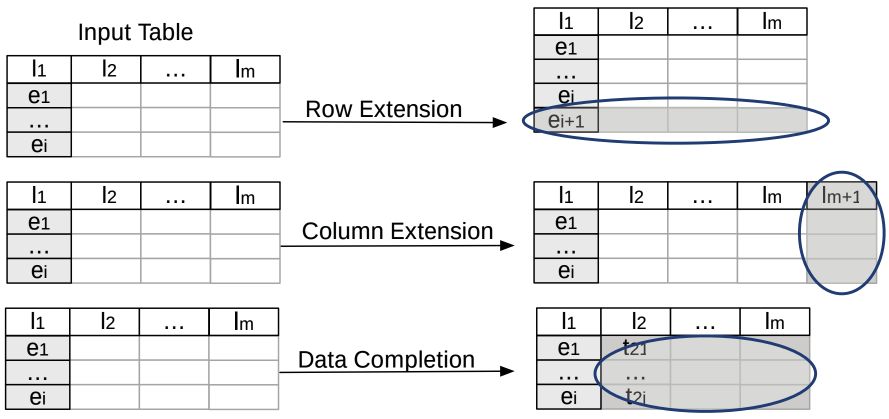


Figure 2.5: Illustration of three table augmentation tasks: row extension, column extension, and data completion.

(KV) that fuses different information sources. For web tables, Dong et al. [2014] firstly identify the relation that is expressed in a table column by checking the column's entities, and reason about which predicate each column could correspond to. This latter task is approached using a standard schema matching method [Venetis et al., 2011], with Freebase as the knowledge base. The extracted relation, together with relational data from other sources, is converted into RDF triples, along with associated confidence scores. The confidence scores are computed based on a graph-based method. Specifically, the triples are fused by machine learning methods from multiple sources, including an existing knowledge base, (i.e., Freebase) and web tables. Consequently, 1.6B triples are generated, of which 324M have a confidence score above 0.7 and 271M have a confidence score above 0.9.

2.7 Table Augmentation

Table augmentation refers to the task of extending a seed table with more data. Specifically, we discuss three tasks in this section: row extension (Sect. 2.7.1), column extension (Sect. 2.7.2), and data completion (Sect. 2.7.3). See Fig. 2.5 for an illustration. One might envisage these functionalities being offered by an intelligent agent that aims to provide assistance for people working with tables (cf. Appendix A).

2.7.1 Row Extension

Row extension aims to extend a given table with more rows or row elements (see Fig. 2.6). It mainly focuses on a particular type of table, namely, relational tables. More specifically, row extension primarily targets horizontal

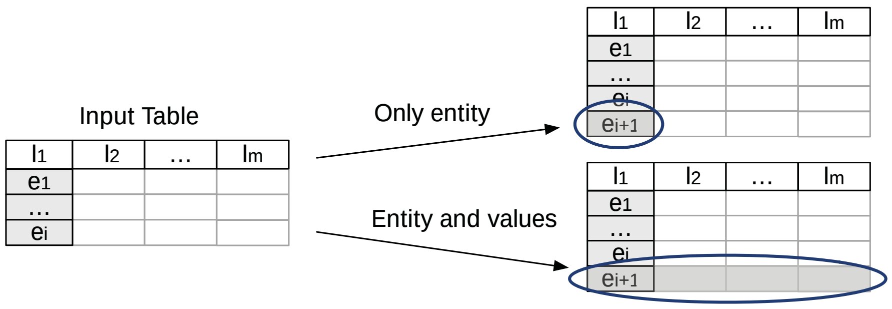


Figure 2.6: Illustration of row extension by adding (Left) only an entity and (Right) an entire row (entity and cell values).

Table 2.11: Overview of row population methods. Notice that table search is inherently involved.

	Data		Tasks	
Reference	**KB**	**Tables**	**Table search**	**Row population**
Wang et al. [2015a]		✓	✓	✓*
Das Sarma et al. [2012]		✓	✓	
Yakout et al. [2012]		✓	✓	✓
Zhang and Balog [2017c]	✓	✓	✓	✓

* Originally developed for concept expansion, but can be used for row population.

relational tables, where rows represent entities and columns describe the attributes of those entities. In such tables there usually exists a *core column* (or *key column*) containing mostly entities [Bhagavatula et al., 2015, Venetis et al., 2011]. Instead of directly providing a complete tuple (row), existing work has focused on identifying entities for populating such core columns (i.e., the Upper scenario in Fig. 2.6). Table 2.11 provides an overview of methods that will be covered below. As we shall see, table search is inherently involved here.

Populating entities in the core column of a table is similar to the problem of *concept expansion*, also known as *entity set expansion*, where a given set of seed entities is to be completed with additional entities [Bron et al., 2013, He and Xin, 2011, Metzger et al., 2013, 2014]. Existing methods for concept expansion suffer from two main issues: input ambiguity and semantic drift (i.e., entities belonging to different concepts are mixed during expansion). Motivated by the intuition that tables tend to group entities that belong to a coherent concept, Wang et al. [2015a] leverage web tables for the concept expansion task, thereby aiming to prevent semantic drift. They provide both the seed entities as well as a concept name are as input. First, they retrieve tables related to the seed entities. Then, they use a graph-based ranking method to rank candidate entities that co-occur with the seed entities in those tables. Specifically, they first expand the set by iteratively adding the most relevant tables based on concept likelihood, and collecting entities there. Then, they refine the earlier estimation and remove less relevant tables based on more complete information. Wang et al. [2015a] find that adding an input concept can address the semantic drift problem for tail concepts. While this method is developed for concept expansion, it is directly applicable to the problem of populating entities in a core column.

Das Sarma et al. [2012] search for *entity complement* tables that are semantically related to entities in the input table (as we have already discussed in Sect. 2.4.2). Then, the top-*k* related tables are used for populating the input table. Das Sarma et al. [2012], however, stop at the table search task. A similar approach is taken in InfoGather [Yakout et al., 2012], where this task is referred to as the *augmentation by example* operation. There, they first search for related tables (cf. Sect. 2.4.2), and then consider entities from these tables, weighted by the table relatedness scores. Yakout et al. [2012] build a schema matching graph among web tables (SMW graph) based on pairwise table similarity. Despite the use of scalable techniques, this remains to be computationally very expensive, which is a main limitation of the approach. Instead of relying only on related tables from a table corpus, Zhang and Balog [2017c] also consider a knowledge base (DBpedia) for identifying candidate entities. Specifically, they collect entities sharing the same types or categories with the input entities from DBpedia, and entities from similar tables (i.e., tables sharing seed entities, having similar captions, or including the same headings) as candidates. They find that entity type information in DBpedia is too general to help identify relevant candidates, and end up using only category information when extracting candidates from DBpedia. It is also shown that using related tables and using a knowledge base are complementary when identifying candidate entities. They develop a generative probabilistic model for the subsequent ranking of candidate entities based on their similarity to (i) other entities in the table, (ii) column headings, and (iii) the caption of the input table. Among the three table elements, seed entities are the most important component for entity ranking, followed by table headings and caption. A combination of the three table elements performs the best in the end. In recent work, Deng et al. [2019] utilize Word2vec to train table embeddings for core column entities. Combining the embedding-based similarity scores with the probability-based scores from [Zhang and Balog, 2017c] results in further performance improvements.

2.7.2 Column Extension

The most widely studied subtask in table augmentation is *column extension*: extending a table with additional columns. This task roughly corresponds

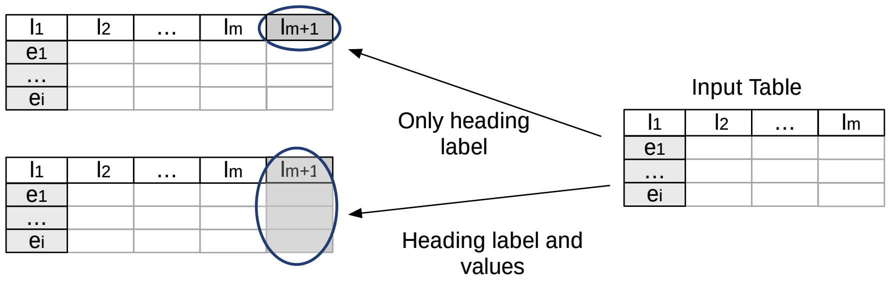


Figure 2.7: Illustration of column extension by adding (Left) only a heading label (Right) an entire column (heading label and cell values).

Table 2.12: Overview of column extension methods.

Reference	Task	Data: Web tables	Data: WP tables	Data: KBs
Cafarella et al. [2008a]	Schema auto-completion	✓		
Das Sarma et al. [2012]	Schema complement	✓		
Lehmberg et al. [2015]	Search join	✓	✓	✓
Zhang and Balog [2017c]	Column population		✓	
Bhagavatula et al. [2013]	Relevant join		✓	

Reference	Task	Output: T_H	Output: $T_H + T_{[:,j]}$
Cafarella et al. [2008a]	Schema auto-completion	✓	
Das Sarma et al. [2012]	Schema complement		
Lehmberg et al. [2015]	Search join		✓
Zhang and Balog [2017c]	Column population	✓	
Bhagavatula et al. [2013]	Relevant join		✓

to the *join* operation in databases. In this context, the set of column heading labels is also often referred to as the table *schema*. Commonly, column extension is approached by first locating similar tables and then considering the column headings/values in those tables. Table 2.12 provides an overview of the methods discussed in this section.

One particular variant of column extension aims to identify additional column heading labels (see Fig. 2.7 (Upper)). As table columns often correspond to entity attributes, this task is also referred to as *attribute discovery* [Yakout et al., 2012] or *schema auto-complete* Cafarella et al. [2008a]. The WebTables system [Cafarella et al., 2008a] implements this functionality based on the *attribute correlation statistics database* (ACSDb). ACSDb contains frequency statistics of attributes and co-occurring attribute pairs in a table corpus. ACSDb comprises 5.4M unique attribute names and 2.6M unique schemas. With these statistics at hand, the next probable attribute can be chosen using a greedy algorithm. The statistics-based method in [Cafarella et al., 2008a] was the first approach to column extension, and was found to be able to provide coherent heading suggestions. However, later research has proven that considering additional features can further improve performance. Das Sarma et al. [2012] focus on finding related tables, with the aim of schema complement. For ranking tables, they consider two factors: (i) the coverage of entities, and (ii) the potential benefits of adding additional attributes from those tables (we discussed the table search method in Sect. 2.4.2). Again, they stop at the table search task. The task of identifying potential attributes or column labels is also known as *schema matching* [Lehmberg et al., 2015] or *column population* [Zhang and Balog, 2017c]. Zhang and Balog [2017c] try to find the headings that can be placed as the next column in an input table. They first find candidate headings from similar tables (the same strategy that they also use for row population). Zhang and Balog [2017c] observe that input entities and table caption contribute comparably to the identification of relevant candidates, while table headings are the least important component. However, similar to row population, all these sources are complementary, i.e., each source can identify candidate headings that none of the others could. In a subsequent ranking step, the candidates are scored based on table similarity, by aggregating element-wise similarities between (corresponding elements of) the input table and related tables. In [Deng et al., 2019], they utilize Word2vec to train embeddings for table headings. Similar to row population, combining the embedding similarity scores with the probabilities from [Zhang and Balog, 2017c] yields further performance improvements. The above approaches differ in what they use as input, i.e., whether they use only table headings [Cafarella et al., 2008a, Lehmberg et al., 2015] or the entire table [Das Sarma et al., 2012, Zhang and Balog, 2017c].

Another variant attempts to augment the input table with entire columns, that is, including both the heading label as well as the corresponding cell values for each row within that column (see Fig. 2.7 (Lower)). Bhagavatula et al. [2013] present the *relevant join* task, which returns a ranked list of column triplets for a given input table. Each triplet consists of *SourceColumn*, *MatchedColumn*, and *CandidateColumn*. *SourceColumn* is from the query table, while *MatchedColumn* and *CandidateColumn* are from the candidate tables. They propose a semantic relatedness measure to find candidate tables from related Wikipedia pages, where page relatedness is estimated based on in-link intersections. Their idea is to compute similarity between columns, such that if *SourceColumn* and *MatchedColumn* share largely similar values, then the input table may be extended with *CandidateColumn*. These candidate columns are classified as relevant or non-relevant, using a linear ranking model, before performing the actual join. To reduce the number of candidate columns, some are filtered out in a pre-processing stage using simple heuristics. Columns that are kept are required to be non-numeric, have more

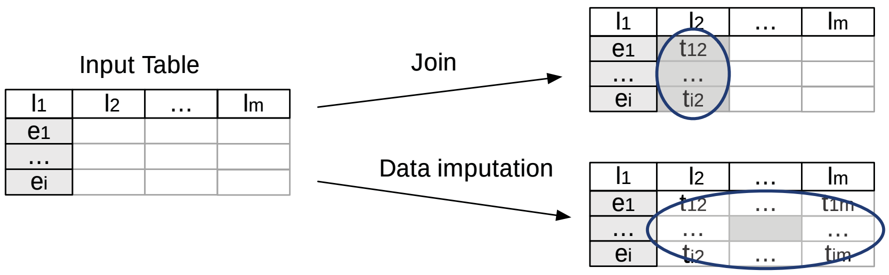


Figure 2.8: Illustration of data completion tasks: (Left) join and (Right) data imputation.

Table 2.13: Overview of data completion methods.

Reference	Data: Tables	Data: Web	Output: $T_{[:,j]}$	Output: $T_{[i,j]}$
Yakout et al. [2012]	✓		✓	
Zhang and Chakrabarti [2013]	✓		✓	
Cafarella et al. [2009]	✓		✓	
Ahmadov et al. [2015b]	✓	✓	✓	✓

than four rows, and an average string length larger than four. Bhagavatula et al. [2013] find that columns containing numeric data make more relevant additions than non-numeric ones. Additionally, more distinct values in the *SourceColumn* and a higher match percentage lead to better quality joins. The join operation is also supported by the Mannheim Search Join Engine [Lehmberg et al., 2015]. It first searches for related tables based on column headings (cf. Sec. 2.4.2), then applies a series of left outer joins between the query table and the returned tables. Afterwards, a consolidation operation is performed to combine attributes. Specifically, they employ a matching operator that relies on data from knowledge bases. Given two columns, similar match (Levenshtein distance) and exact match are used for matching headings. Lehmberg et al. [2015] observe that similar match returns on average 3.4 times more tables than exact match. Among different table corpora, web tables provide the largest number of relevant tables, and Wikipedia tables tend to be populous on certain topics, such as countries and films.

2.7.3 Data Completion

Data completion for tables refers to the task of filling in the empty table cells. Table 2.13 summarizes the methods we discuss here. One variant of this task attempts to find the cell values for an entire column (see Fig. 2.8 (Upper)). This is known as the *augmentation by attribute name* operation in the InfoGather system [Yakout et al., 2012]. This is typical of a scenario where the core entity column as well as the column headings are given in a relational table, and the values for the corresponding attributes (*augmenting attributes*) are to be filled in. The system in [Yakout et al., 2012] takes the incomplete table as input to search for matching tables, then extracts attribute values from those tables. It is worthwhile to point out that InfoGather focuses on finding values that are entities. An extended version of the system, InfoGather+ [Zhang and Chakrabarti, 2013], focuses on numerical and time-varying attributes. They use undirected graphical models and build a semantic graph that labels columns with units, scales, and timestamps, and computes semantic matches between columns. Their experiments are conducted on three types of tables: company (revenue and profit), country (area and tax rate) and city (population). Zhang and Chakrabarti [2013] find that the conversion rules (manually designed unit conversion mapping) achieve higher coverage than string-based schema matching methods. Similar to InfoGather's *augmentation by attribute name* operation, the *extend* operation in the OCTOPUS systems [Cafarella et al., 2009] enables the user to add more columns to a table by performing a join. It takes a keyword query and a given (existing) table column as input, where the keyword describes the newly added column. Different from a regular join, the added column is not necessarily an existing column. It may be formed row-by-row by combining information from multiple related tables (see Sect. 2.4.1 for the table search operation). However, Cafarella et al. [2009] rely on simple methods like edit-distance for schema matching, which leaves room for improvement.

Another flavor of the data completion task focuses on filling in missing values for individual cells, referred to as *data imputation* (see Fig. 2.8 (Lower)). Ahmadov et al. [2015b] present a hybrid imputation method that combines a lookup-based approach, based on a corpus of web tables, and a model-based approach that uses machine learning (e.g., k-nearest neighbors or linear regression) to predict the value for a missing cell. It is worth noting that all the above methods rely only on tables and ignore the cases where no similar tables can be found. The method in [Ahmadov et al., 2015b] is shown to be able to improve coverage. However, being able to automatically decide when to do simple lookup and when to employ a machine learned model remains an open challenge.

2.8 Summary and Conclusions

Tables are a powerful and popular tool for organizing and manipulating data. Research on web tables has seen nearly two decades of development. During this long period, the research focus has evolved considerably, from low level tasks (table extraction) to tapping more and more into the actual knowledge contained in tables (for search and for augmenting existing knowledge bases). In this chapter, we have reviewed past progress and identified open research issues, organized around six main tasks.

In this thesis, we address a number of tasks, which include table search, table generation, and table completion. For table search, previous work focuses on lexical matching, while we improve the keyword table search task with semantic matching and present the query-by-table paradigm. We propose table generation models for free text (natural language) queries. For table augmentation, previous work only utilizes other tables, while we additionally incorporate a knowledge base. More specific differences will be discussed throughout the respective technical chapters, Chapters 3-7.

Chapter 3

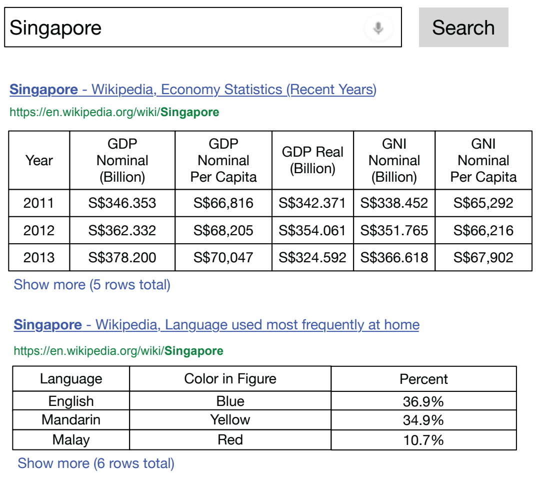
Keyword Table Search

In this chapter, we introduce and address the problem of ad hoc table retrieval: answering a keyword query with a ranked list of tables. See Fig. 3.1 for an illustration. Tables can provide direct answers to many information needs, and can effectively summarize, e.g., properties of entities, thereby saving users the efforts of looking these up one by one. Thus this task is not only interesting on its own account, but is also being used as a core component in many other table-based information access scenarios, such as table completion (Chapter 5) or table generation (Chapter 6). The main novel contribution of this chapter is a method for performing semantic matching between queries and tables in order to fix the "vocabulary gap" problem. Specifically, we (i) represent queries and tables in multiple semantic spaces (both discrete sparse and continuous dense vector representations) and (ii) introduce various similarity measures for matching those semantic representations. We consider all possible combinations of semantic representations and similarity measures and use these as features in a supervised learning model. Using a purpose-built test collection based on Wikipedia tables, we demonstrate significant and substantial improvements over a state-of-the-art baseline.

The chapter is organized as follows. We introduce and formalize the ad hoc table ranking task, and present both unsupervised and supervised baseline approaches in Sect. 3.1. We present a set of novel semantic matching methods that go beyond lexical similarity in Sect. 3.2. We develop a standard test collection for this task in Sect. 3.3 and demonstrate the effectiveness of our approaches in Sect. 3.4. Section 3.5 concludes this chapter.

3.1 Ad Hoc Table Retrieval

We formalize the ad hoc table retrieval task, explain what information is associated with a table, and introduce baseline methods.



Singapore - Wikipedia, Economy Statistics (Recent Years)
https://en.wikipedia.org/wiki/Singapore

Year	GDP Nominal (Billion)	GDP Nominal Per Capita	GDP Real (Billion)	GNI Nominal (Billion)	GNI Nominal Per Capita
2011	S$346.353	S$66,816	S$342.371	S$338.452	S$65,292
2012	S$362.332	S$68,205	S$354.061	S$351.765	S$66,216
2013	S$378.200	S$70,047	S$324.592	S$366.618	S$67,902

Show more (5 rows total)

Singapore - Wikipedia, Language used most frequently at home
https://en.wikipedia.org/wiki/Singapore

Language	Color in Figure	Percent
English	Blue	36.9%
Mandarin	Yellow	34.9%
Malay	Red	10.7%

Show more (6 rows total)

Figure 3.1: Ad hoc table retrieval: given a keyword query, the system returns a ranked list of tables.

Table 3.1: Notation used in this chapter.

Symbol	Description
q	Keyword query
T	Candidate table
C	Table collection

3.1.1 Problem Statement

Given a keyword query q, *ad hoc table retrieval* is the task of returning a ranked list of tables, $(T_1, \ldots, T_k)$, from a collection of tables C. Being an ad hoc task, the relevance of each returned table T_i is assessed independently of all other returned tables $T_j, i \neq j$. Hence, the ranking of tables boils down to the problem of assigning a score to each table in the corpus: $score(q, T)$. Tables are then sorted in descending order of their scores. We list the notation used in this chapter in Table 3.1.

3.1.2 Unsupervised Ranking

An easy and straightforward way to perform the table ranking task is by adopting standard document ranking methods. Cafarella et al. [2009, 2008a] utilize web search engines to retrieve relevant documents; tables are then extracted from the highest-ranked documents. Rather than relying on external services, we represent tables as either single- or multi-field documents and apply standard document retrieval techniques.

Table 3.2: Baseline features for table retrieval I.

Query features		**Source**	**Value**
QLEN	Number of query terms	[Tyree et al., 2011]	{1,...,n}
IDF$_f$	Sum of query IDF scores in field f	[Qin et al., 2010]	$[0, \infty)$
Table features			
#rows	The number of rows in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]	{1,...,n}
#cols	The number of columns in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]	{1,...,n}
#of NULLs in table	The number of empty table cells	[Cafarella et al., 2008a, Bhagavatula et al., 2013]	{0,...,n}
PMI	The ACSDb-based schema coherency score	[Cafarella et al., 2008a]	$(-\infty, \infty)$
inLinks	Number of in-links to the page embedding the table	[Bhagavatula et al., 2013]	{0,...,n}
outLinks	Number of out-links from the page embedding the table	[Bhagavatula et al., 2013]	{0,...,n}
pageViews	Number of page views	[Bhagavatula et al., 2013]	{0,...,n}
tableImportance	Inverse of number of tables on the page	[Bhagavatula et al., 2013]	$(0, 1]$
tablePageFraction	Ratio of table size to page size	[Bhagavatula et al., 2013]	$(0, 1]$

Single-field Document Representation

In the simplest case, all text associated with a given table is used as the table's representation. This representation is then scored using existing retrieval methods, such as BM25 or language models.

Multi-field Document Representation

Rather than collapsing all textual content into a single-field document, it may be organized into multiple fields, such as table caption, table headers, table body, etc. (cf. Table. 2.2). For multi-field ranking, Pimplikar and Sarawagi [2012] employ a *late fusion* strategy [Zhang and Balog, 2017a]. That is, each field is scored independently against the query, then a weighted sum of the field-level similarity scores is taken:

$$score(q, T) = \sum_i w_i \times score(q, f_i) \ , \tag{3.1}$$

where f_i denotes the ith (document) field for table T and w_i is the corresponding field weight (such that $\sum_i w_i = 1$). $score(q, f_i)$ may be computed using any standard retrieval method. We use language models in our experiments.

Table 3.3: Baseline features for table retrieval II.

Query features		**Source**	**Value**
#hitsLC	Total query term frequency in the leftmost column cells	[Cafarella et al., 2008a]	{0,...,n}
#hitsSLC	Total query term frequency in second-to-leftmost column cells	[Cafarella et al., 2008a]	{0,...,n}
#hitsB	Total query term frequency in the table body	[Cafarella et al., 2008a]	{0,...,n}
qInPgTitle	Ratio of the number of query tokens found in page title to total number of tokens	[Bhagavatula et al., 2013]	$[0,1]$
qInTableTitle	Ratio of the number of query tokens found in table title to total number of tokens	[Bhagavatula et al., 2013]	$[0,1]$
yRank	Rank of the table's Wikipedia page in Web search engine results for the query	[Bhagavatula et al., 2013]	{1,...,n}
MLM similarity	Language modeling score between query and multi-field document repr. of the table	[Chen et al., 2016]	$(-\infty,0)$

3.1.3 Supervised Ranking

The state-of-the-art in document retrieval (and in many other retrieval tasks) is to employ supervised learning [Liu, 2011]. Features may be categorized into three groups: (i) document, (ii) query, and (iii) query-document features [Qin et al., 2010]. Analogously, we distinguish between three types of features: (i) table, (ii) query, and (iii) query-table features. In Table 3.2 and Table 3.3, we summarize the features from previous work on table search [Cafarella et al., 2008a, Bhagavatula et al., 2013]. We also include a number of additional features that have been used in other retrieval tasks, such as document and entity ranking; we do not regard these as novel contributions.

Query Features

Query features have been shown to improve retrieval performance for document ranking [Macdonald et al., 2012]. We adopt two query features from document retrieval, namely, the number of terms in the query [Tyree et al., 2011], and query IDF [Qin et al., 2010] according to: $IDF_f(q) = \sum_{t \in q} IDF_f(t)$, where $IDF_f(t)$ is the IDF score of term t in field f. This feature is computed for the following fields: page title, section title, table caption, table heading, table body, and "catch-all" (the concatenation of all textual content in the table).

Table Features

Table features depend only on the table itself and aim to reflect the quality of the given table (irrespective of the query). Some features are simple characteristics, like the number of rows, columns, and empty cells [Cafarella et al., 2008a, Bhagavatula et al., 2013]. One important feature is Point-wise Mutual Information (PMI), which is taken from linguistics research, and expresses the coherency of a table. The correlation between two table headings cells, h_i and h_j, is given by: $PMI(h_i, h_j) = \log\big(P(h_i, h_j)/(P(h_i)P(h_j))\big)$. A table's PMI is computed by calculating the PMI values between all pairs of column headings of that table, and then taking their average. Following Cafarella et al. [2008a], we compute PMI by obtaining frequency statistics from the Attribute Correlation Statistics Database (ACSDb) [Cafarella et al., 2008b], which contains table heading information derived from millions of tables extracted from a large web crawl.

Another group of features has to do with the page that embeds the table, by considering its connectivity (inLinks and outLinks), popularity (pageViews), and the table's importance within the page (tableImportance and tablePageFraction).

Query-Table Features

Features in the last group express the degree of matching between the query and a given table. This matching may be based on occurrences of query terms in the page title (qInPgTitle) or in the table caption (qInTableTitle). Alternatively, it may be based on specific parts of the table, such as the leftmost column (#hitsLC), second-to-left column (#hitsSLC), or table body (#hitsB). Tables are typically embedded in (web) pages. The rank at which a table's parent page is retrieved by an external search engine is also used as a feature (yRank). (In our experiments, we use the Wikipedia search API to obtain this ranking.) Furthermore, we take the Mixture of Language Models (MLM) similarity score [Ogilvie and Callan, 2003] as a feature, which is actually the best performing method among the four text-based baseline methods (cf. Sect. 3.4). Importantly, all these features are based on lexical matching. Our goal in this chapter is to also enable semantic matching; this is what we shall discuss in the next section.

3.2 Semantic Matching

This section presents our main contribution, which is a set of novel semantic matching methods for table retrieval. The main idea is to go beyond lexical matching by representing both queries and tables in some semantic space, and measuring the similarity of those semantic (vector) representations. Our

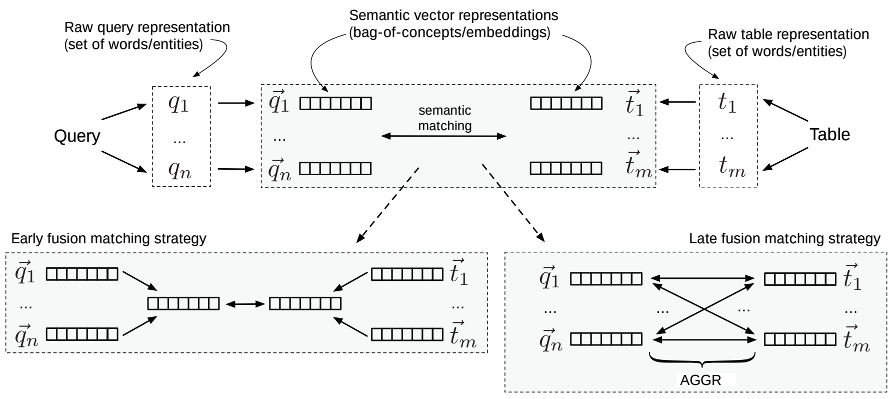

Raw query representation (set of words/entities)
Semantic vector representations (bag-of-concepts/embeddings)
Raw table representation (set of words/entities)
Query
Table
semantic matching
Early fusion matching strategy
Late fusion matching strategy
AGGR

Figure 3.2: Our methods for computing query-table similarity using semantic representations.

approach consists of three main steps, which are illustrated in Figure 3.2. These are as follows (moving from outwards to inwards on the figure):

1. The "raw" content of a query/table is represented as a set of terms, where terms can be either words or entities (Sect. 3.2.1).
2. Each of the raw terms is mapped to a semantic vector representation (Sect. 3.2.2).
3. The semantic similarity (matching score) between a query-table pair is computed based on their semantic vector representations (Sect. 3.2.3).

We compute query-table similarity using all possible combinations of semantic representations and similarity measures, and use the resulting semantic similarity scores as features in a learning-to-rank approach. Table 3.4 summarizes these features.

3.2.1 Content Extraction

We represent the "raw" content of the query/table as a set of terms, where terms can be either words (string tokens) or entities (from a knowledge base). We denote these as $\{q_1, \ldots, q_n\}$ and $\{t_1, \ldots, t_m\}$ for query q and table T, respectively.

Word-based

It is a natural choice to simply use word tokens to represent query/table content. That is, $\{q_1, \ldots, q_n\}$ is comprised of the unique words in the query. As for the table, we let $\{t_1, \ldots, t_m\}$ contain all unique words from the title, caption, and headings of the table. Mind that at this stage we are only considering the presence/absence of words. During the query-table similarity matching, the importance of the words will also be taken into account (Sect. 3.2.3).

Entity-based

Many tables are focused on specific entities [Zhang and Balog, 2017c]. Therefore, considering the entities contained in a table amounts to a meaningful representation of its content. We use the DBpedia knowledge base as our entity repository. Since we work with tables extracted from Wikipedia, the entity annotations are readily available (otherwise, entity annotations could be obtained automatically, see, e.g., [Venetis et al., 2011]). Importantly, instead of blindly including all entities mentioned in the table, we wish to focus on salient entities. It has been observed in prior work [Venetis et al., 2011, Bhagavatula et al., 2015] that tables often have a *core column*, containing mostly entities, while the rest of the columns contain properties of these entities (many of which are entities themselves). We write E_{cc} to denote the set of entities that are contained in the core column of the table, and describe our core column detection method below. In addition to the entities taken directly from the body part of the table, we also include entities that are related to the page title (T_{pt}) and to the table caption (T_{tc}). We obtain those by using the page title and the table caption, respectively, to retrieve relevant entities from the knowledge base. We write $R_k(s)$ to denote the set of top-k entities retrieved for the query s. We detail the entity ranking method in Sect. 3.2.1. Finally, the table is represented as the union of three sets of entities, originating from the core column, page title, and table caption: $\{t_1, \ldots, t_m\} = E_{cc} \cup R_k(T_{pt}) \cup R_k(T_{tc})$.

To get an entity-based representation for the query, we issue the query against a knowledge base to retrieve relevant entities, using the same retrieval method as above. I.e., $\{q_1, \ldots, q_n\} = R_k(q)$.

Core Column Detection

We introduce a simple and effective core column detection method. It is based on the notion of *column entity rate*, which is defined as the ratio of cells in a column that contain an entity. We write $cer(T_{c[j]})$ to denote the column entity rate of column j in table T. Then, the index of the core column becomes: $\arg\max_{j=1..T_{|c|}} cer(T_{c[j]})$, where $T_{|c|}$ is the number of columns in T.

Entity Retrieval

We employ a fielded entity representation with five fields (names, categories, attributes, similar entity names, and related entity names) and rank entities using the Mixture of Language Models approach [Ogilvie and Callan, 2003].

Table 3.4: Semantic similarity features. Each row represents 4 features (one for each similarity matching method, cf. Table 3.5). All features are in $[-1, 1]$.

Features	Semantic repr.	Raw repr.
Entity_*	Bag-of-entities	entities
Category_*	Bag-of-categories	entities
Word_*	Word embeddings	words
Graph_*	Graph embeddings	entities

The field weights are set uniformly. This corresponds to the MLM-all model in [Hasibi et al., 2017b] and is shown to be a solid baseline. We return the top-k entities, where k is set to 10.

3.2.2 Semantic Representations

Next, we embed the query/table terms in a semantic space. That is, we map each table term t_i to a vector representation $\vec{t}_i$, where $\vec{t}_i[j]$ refers to the jth element of that vector. For queries, the process goes analogously. We discuss two main kinds of semantic spaces, bag-of-concepts and embeddings, with two alternatives within each. The former uses sparse and discrete, while the latter employs dense and continuous-valued vectors. A particularly nice property of our semantic matching framework is that it allows us to deal with these two different types of representations in a unified way.

Bag-of-concepts

One alternative for moving from the lexical to the semantic space is to represent tables/queries using specific concepts. In this work, we use entities and categories from a knowledge base. These two semantic spaces have been used in the past for various retrieval tasks, in duet with the traditional bag-of-words content representation. For example, entity-based representations have been used for document retrieval [Xiong et al., 2017, Raviv et al., 2016] and category-based representations have been used for entity retrieval [Balog et al., 2011]. One important difference from previous work is that instead of representing the entire query/table using a single semantic vector, we map each individual query/table term to a separate semantic vector, thereby obtaining a richer representation.

We use the entity-based raw representation from the previous section, that is, t_i and q_j are specific entities. Below, we explain how table terms t_j are projected to $\vec{t}_i$, which is a sparse discrete vector in the entity/category space; for query terms it follows analogously.

Bag-of-entities Each element in $\vec{t}_i$ corresponds to a unique entity. Thus, the dimensionality of $\vec{t}_i$ is the number of entities in the knowledge base (on the order of millions). $\vec{t}_i[j]$ has a value of 1 if entities i and j are related (there exists a link between them in the knowledge base), and 0 otherwise.

Bag-of-categories Each element in $\vec{t}_i$ corresponds to a Wikipedia category. Thus, the dimensionality of $\vec{t}_i$ amounts to the number of Wikipedia categories (on the order hundreds of thousands). The value of $\vec{t}_i[j]$ is 1 if entity i is assigned to Wikipedia category j, and 0 otherwise.

Embeddings

Recently, unsupervised representation learning methods have been proposed for obtaining embeddings that predict a distributional context, i.e., word embeddings [Mikolov et al., 2013, Pennington et al., 2014] or graph embeddings [Perozzi et al., 2014, Tang et al., 2015, Ristoski and Paulheim, 2016]. Such vector representations have been utilized successfully in a range of IR tasks, including ad hoc retrieval Ganguly et al. [2015], Mitra et al. [2016], contextual suggestion Manotumruksa et al. [2016], cross-lingual IR Vulić and Moens [2015], community question answering Zhou et al. [2015], short text similarity Kenter and de Rijke [2015], and sponsored search Grbovic et al. [2015]. We consider both word-based and entity-based raw representations from the previous section and use the corresponding (pre-trained) embeddings as follows.

Word embeddings We map each query/table word to a word embedding. Specifically, we use word2vec [Mikolov et al., 2013] with 300 dimensions, trained on Google News data.

Graph embeddings We map each query/table entity to a graph embedding. In particular, we use RDF2vec [Ristoski and Paulheim, 2016] with 200 dimensions, trained on DBpedia 2015-10.

3.2.3 Similarity Measures

The final step is concerned with the computation of the similarity between a query-table pair, based on the semantic vector representations we have obtained for them. We introduce two main strategies, which yield four specific similarity measures. These are summarized in Table 3.5.

Early Fusion

The first idea is to represent the query and the table each with a single vector. Their similarity can then simply be expressed as the similarity of the

Table 3.5: Similarity measures.

Measure	Equation
Early	$\cos(\vec{C}_q, \vec{C}_T)$
Late-max	$\max(\{\cos(\vec{q}_i, \vec{t}_j) : i \in [1..n], j \in [1..m]\})$
Late-sum	$\text{sum}(\{\cos(\vec{q}_i, \vec{t}_j) : i \in [1..n], j \in [1..m]\})$
Late-avg	$\text{avg}(\{\cos(\vec{q}_i, \vec{t}_j) : i \in [1..n], j \in [1..m]\})$

corresponding vectors. We let $\vec{C}_q$ be the centroid of the query term vectors ($\vec{C}_q = \sum_{i=1}^{n} \vec{q}_i / n$). Similarly, $\vec{C}_T$ denotes the centroid of the table term vectors. The query-table similarity is then computed by taking the cosine similarity of the centroid vectors. When query/table content is represented in terms of words, we additionally make use of word importance by employing standard TF-IDF term weighting. Note that this only applies to word embeddings (as the other three semantic representations are based on entities). In case of word embeddings, the centroid vectors are calculated as $\vec{C}_T = \sum_{i=1}^{m} \vec{t}_i \times TFIDF(t_i)$. The computation of $\vec{C}_q$ follows analogously.

Late Fusion

Instead of combining all semantic vectors q_i and t_j into a single one, late fusion computes the pairwise similarity between all query and table vectors first, and then aggregates those. We let S be a set that holds all pairwise cosine similarity scores: $S = \{\cos(\vec{q}_i, \vec{t}_j) : i \in [1..n], j \in [1..m]\}$. The query-table similarity score is then computed as $\text{aggr}(S)$, where $\text{aggr}()$ is an aggregation function. Specifically, we use $\max()$, $\text{sum}()$ and $\text{avg}()$ as aggregators; see the last three rows in Table 3.5 for the equations.

3.3 Test Collection

We introduce our test collection, including the table corpus, test and development query sets, and the procedure used for obtaining relevance assessments.

3.3.1 Table Corpus

We use the WikiTables corpus [Bhagavatula et al., 2015], which comprises 1.6M tables extracted from Wikipedia (dump date: 2015 October). The following information is provided for each table: table caption, column headings, table body, (Wikipedia) page title, section title, and table statistics like number of headings rows, columns, and data rows. We further replace all links in the table body with entity identifiers from the DBpedia knowledge

Table 3.6: Example queries from our query set.

Queries from [Cafarella et al., 2009]	**Queries from [Venetis et al., 2011]**
video games	asian coutries currency
us cities	laptops cpu
kings of africa	food calories
economy gdp	guitars manufacturer
fifa world cup winners	clothes brand

base (version 2015-10) as follows. For each cell that contains a hyperlink, we check if it points to an entity that is present in DBpedia. If yes, we use the DBpedia identifier of the linked entity as the cell's content; otherwise, we replace the link with the anchor text, i.e., treat it as a string.

3.3.2 Queries

We sample a total of 60 test queries from two independent sources (30 from each): (1) *Query subset 1 (QS-1)*: Cafarella et al. [2009] collected 51 queries from Web users via crowdsourcing (using Amazon's Mechanical Turk platform, users were asked to suggest topics or supply URLs for a useful data table). (2) *Query subset 2 (QS-2)*: Venetis et al. [2011] analyzed the query logs from Google Squared (a service in which users search for structured data) and constructed 100 queries, all of which are a combination of an instance class (e.g., "laptops") and a property (e.g., "cpu"). Following [Bhagavatula et al., 2013], we concatenate the class and property fields into a single query string (e.g., "laptops cpu"). Table 3.6 lists some examples.

3.3.3 Relevance Assessments

We collect graded relevance assessments by employing three independent (trained) judges. For each query, we pool the top 20 results from five baseline methods (cf. Sect. 3.4.3), using default parameter settings. (Then, we train the parameters of those methods with help of the obtained relevance labels.) Each query-table pair is judged on a three point scale: 0 (non-relevant), 1 (somewhat relevant), and 2 (highly relevant). Annotators were situated in a scenario where they need to create a table on the topic of the query, and wish to find relevant tables that can aid them in completing that task. Specifically, they were given the following labeling guidelines: (i) a table is *non-relevant* if it is unclear what it is about (e.g., misses headings or caption) or is about a different topic; (ii) a table is *relevant* if some cells or values could be used from this table; and (iii) a table is *highly relevant* if large blocks or several values could be used from it when creating a new table on the query topic.

Table 3.7: Table retrieval evaluation results.

Method	NDCG@5	NDCG@10	NDCG@15	NDCG@20
Single-field document ranking	0.4315	0.4344	0.4586	0.5254
Multi-field document ranking	0.4770	0.4860	0.5170	0.5473
WebTable [Cafarella et al., 2008a]	0.2831	0.2992	0.3311	0.3726
WikiTable [Bhagavatula et al., 2013]	0.4903	0.4766	0.5062	0.5206
LTR baseline (this chapter)	0.5527	0.5456	0.5738	0.6031
STR (this chapter)	**0.5951**	**0.6293**†	**0.6590**‡	**0.6825**†

We take the majority vote as the relevance label; if no majority agreement is achieved, we take the average of the scores as the final label. To measure inter-annotator agreement, we compute the Kappa test statistics on test annotations, which is 0.47. According to Fleiss et al. [1971], this is considered as moderate agreement. In total, 3120 query-table pairs are annotated as test data. Out of these, 377 are labeled as highly relevant, 474 as relevant, and 2269 as non-relevant.

3.4 Evaluation

In this section, we list our research questions (Sect. 3.4.1), discuss our experimental setup (Sect. 3.4.2), introduce the baselines we compare against (Sect. 3.4.3), and present our results (Sect. 3.4.4) followed by further analysis (Sect. 3.4.5).

3.4.1 Research Questions

The research questions we seek to answer are as follows.

RQ1/A Can semantic matching improve retrieval performance?

RQ1/B Which of the semantic representations is the most effective?

RQ1/C Which of the similarity measures performs better?

3.4.2 Experimental Setup

We evaluate table retrieval performance in terms of Normalized Discounted Cumulative Gain (NDCG) at cut-off points 5, 10, 15, and 20. To test significance, we use a two-tailed paired t-test and write †/‡ to denote significance at the 0.05 and 0.005 levels, respectively.

Our implementations are based on Nordlys [Hasibi et al., 2017a]. Many of our features involve external sources, which we explain below. To compute the entity-related features (i.e., features in Table 3.3 and Table 3.4 as

Table 3.8: Comparison of semantic features, used in combination with baseline features (from Table 3.3 and Table 3.4), in terms of NDCG@20. Relative improvements are shown in parentheses. Statistical significance is tested against the LTR baseline in Table 3.7.

Sem. Repr.	**Early**	**Late-max**	**Late-sum**	**Late-avg**	**ALL**
Bag-of-entities	0.6754 (+11.99%)	0.6407 (+6.23%)†	0.6697 (+11.04%)‡	0.6733 (+11.64%)‡	0.6696 (+11.03%)‡
Bag-of-categories	0.6287 (+4.19%)	0.6245 (+3.55%)	0.6315 (+4.71%)†	0.6240 (+3.47%)	0.6149 (+1.96%)
Word embeddings	0.6181 (+2.49%)	0.6328 (+4.92%)	0.6371 (+5.64%)†	0.6485 (+7.53%)†	0.6588 (+9.24%)†
Graph embeddings	0.6326 (+4.89%)	0.6142 (+1.84%)	0.6223 (+3.18%)	0.6316 (+4.73%)	0.6340 (+5.12%)
ALL	0.6736 (+11.69%)†	0.6631 (+9.95%)†	0.6831 (+13.26%)‡	0.6809 (+12.90%)‡	0.6825 (13.17%)‡

well as the features based on the bag-of-entities and bag-of-categories representations in Table 3.4), we use entities from the DBpedia knowledge base that have an abstract (4.6M in total). The table's Wikipedia rank (yRank) is obtained using Wikipedia's MediaWiki API. The PMI feature is estimated based on the ACSDb corpus [Cafarella et al., 2008b]. For the distributed representations, we take pre-trained embedding vectors, as explained in Sect. 3.2.2.

3.4.3 Baselines

We implement four baseline methods from the literature.

Single-field document ranking In [Cafarella et al., 2009, 2008a] tables are represented and ranked as ordinary documents. Specifically, we use Language Models with Dirichlet smoothing, and optimize the smoothing parameter using a parameter sweep.

Multi-field document ranking Pimplikar and Sarawagi [2012] represent each table as a fielded document, using five fields: Wikipedia page title, table section title, table caption, table body, and table headings. We use the Mixture of Language Models approach [Ogilvie and Callan, 2003] for ranking. Field weights are optimized using the coordinate ascent algorithm; smoothing parameters are trained for each field individually.

WebTable The method by Cafarella et al. [2008a] uses the features in Table 3.3 and Table 3.4 with [Cafarella et al., 2008a] as source. Following [Cafarella et al., 2008a], we train a linear regression model with 5-fold cross-validation.

WikiTable The approach by Bhagavatula et al. [2013] uses the features in Table 3.3 and Table 3.4 with [Bhagavatula et al., 2013] as source. We train a Lasso model with coordinate ascent with 5-fold cross-validation.

Additionally, we introduce a learning-to-rank baseline:

LTR baseline It uses the full set of features listed in Table 3.3 and Table 3.4. We employ pointwise regression using the Random Forest algorithm.[1] We set the number of trees to 1000 and the maximum number of features in each tree to 3. We train the model using 5-fold cross-validation (w.r.t. NDCG@20); reported results are averaged over 5 runs.

The baseline results are presented in the top block of Table 3.7. It can be seen from this table that our LTR baseline (row five) outperforms all existing methods from the literature; the differences are substantial and statistically significant. Therefore, in the remainder of this chapter, we shall compare against this strong baseline, using the same learning algorithm (Random Forests) and parameter settings. We note that our emphasis is on the semantic matching features and not on the supervised learning algorithm.

3.4.4 Experimental Results

The last line of Table 3.7 shows the results for our semantic table retrieval (STR) method. It combines the baseline set of features (Table 3.3 and Table 3.4) with the set of novel semantic matching features (from Table 3.4, 16 in total). We find that these semantic features bring in substantial and statistically significant improvements over the LTR baseline. Thus, we answer RQ1/A positively. The relative improvements range from 7.6% to 15.3%, depending on the rank cut-off.

To answer RQ1/B and RQ1/C, we report on all combinations of semantic representations and similarity measures in Table 3.8. In the interest of space, we only report on NDCG@20; the same trends were observed for other NDCG cut-offs. Cells with a white background show retrieval performance when extending the LTR baseline with a single feature. Cells with a grey background correspond to using a given semantic representation with different similarity measures (rows) or using a given similarity measure with different semantic representations (columns). The first observation is that all features improve over the baseline, albeit not all of these improvements are statistically significant. Concerning the comparison of different semantic representations (RQ1/B), we find that bag-of-entities and word embeddings achieve significant improvements; see the rightmost column of Table 3.8. It is worth pointing out that for word embeddings the four similarity measures seem to complement each other, as their combined performance is better than that of any individual method. It is not the case for bag-of-entities, where only one of the similarity measures (Late-max) is improved by the combination. Overall, in answer to RQ1/B, we find the bag-of-entities representation to be the most effective one. The fact that this

[1]We also experimented with Gradient Boosting regression and Support Vector Regression, and observed the same general patterns regarding feature importance. However, their overall performance was lower than that of Random Forests.

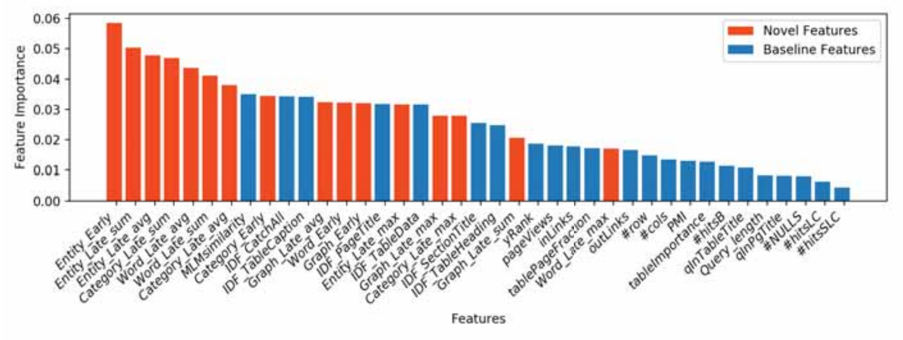


Figure 3.3: Normalized feature importance (measured in terms of Gini score).

sparse representation outperforms word embeddings is regarded as a somewhat surprising finding, given that the latter has been trained on massive amounts of (external) data.

As for the choice of similarity measure (RQ1/C), it is difficult to name a clear winner when a single semantic representation is used. The relative differences between similarity measures are generally small (below 5%). When all four semantic representations are used (bottom row in Table 3.8), we find that Late-sum and Late-avg achieve the highest overall improvement. Importantly, when using all semantic representations, all four similarity measures improve significantly and substantially over the baseline. We further note that the combination of all similarity measures do not yield further improvements over Late-sum or Late-avg. In answer to RQ1/C, we identify the late fusion strategy with sum or avg aggregation (i.e., Late-sum or Late-avg) as the preferred similarity method.

3.4.5 Analysis

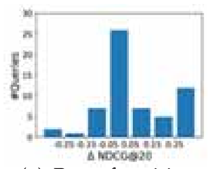
We continue with further analysis of our results.

Features

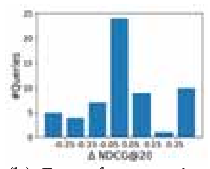
Figure 3.3 shows the importance of individual features for the table retrieval task, measured in terms of Gini importance. The novel features are distinguished by color. We observe that 8 out of the top 10 features are semantic features introduced in this chapter.

Semantic Representations

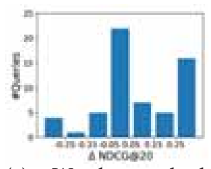
To analyze how the four semantic representations affect retrieval performance on the level of individual queries, we plot the difference between the



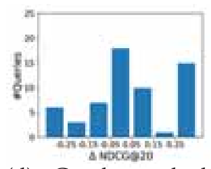
(a) Bag-of-entities



(b) Bag-of-categories



(c) Word embeddings



(d) Graph embeddings

Figure 3.4: Distribution of query-level differences between the LTR baseline and a given semantic representation.

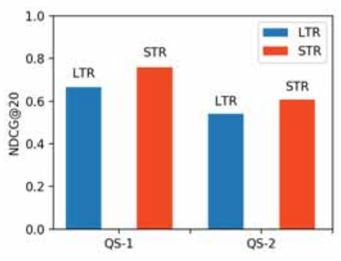
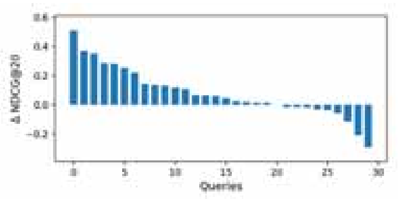


Figure 3.5: Table retrieval results, LTR baseline vs. STR, on the two query subsets in terms of NDCG@20.

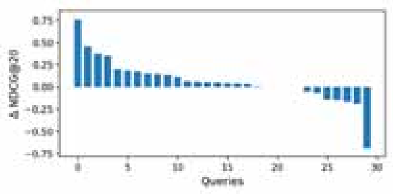
LTR baseline and each semantic representation in Figure 3.4. The histograms show the distribution of queries according to NDCG@20 score difference (Δ): the middle bar represents no change (Δ <0.05), while the leftmost and rightmost bars represents the number of queries that were hurt and helped substantially, respectively (Δ >0.25). We observe similar patterns for the bag-of-entities and word embeddings representations; the former has less queries that were significantly helped or hurt, while the overall improvement (over all topics) is larger. We further note the similarity of the shapes of the distributions for bag-of-categories and graph embeddings.

Query Subsets

On Figure 3.5, we plot the results for the LTR baseline and for our STR method according to the two query subsets, QS-1 and QS-2, in terms of NDCG@20. Generally, both methods perform better on QS-1 than on QS-2. This is mainly because QS-2 queries are more focused (each targeting a specific type of instance, with a required property), and thus are considered more difficult. Importantly, STR achieves consistent improvements over LTR on both query subsets.



(a) QS-1



(b) QS-2

Figure 3.6: Query-level differences on the two query subsets between the LTR baseline and STR. Positive values indicate improvements made by the latter.

Table 3.9: Example queries from our query set. Rel denotes table relevance level. LTR and STR refer to the positions on which the table is returned by the respective method.

Query	Rel	LTR	STR
QS-1-24: *stocks*			
Stocks for the Long Run / Key Data Findings: annual real returns	2	-	6
TOPIX / TOPIX New Index Series	1	9	-
Hang Seng Index / Selection criteria for the HSI constituent stocks	1	-	-
QS-1-21: *ibanez guitars*			
Ibanez / Serial numbers	2	1	2
Corey Taylor / Equipment	1	2	3
Fingerboard / Examples	1	4	5
QS-2-27: *board games number of players*			
List of Japanese board games	1	13	1
List of licensed Risk game boards / Risk Legacy	1	-	3
QS-2-21: *cereals nutritional value*			
Sesame / Sesame seed kernels, toasted	2	1	8
QS-2-20: *irish counties area*			
Counties of Ireland / List of counties	2	2	1
List of Irish counties by area / See also	2	1	2
List of flags of Ireland / Counties of Ireland Flags	2	-	3
Provinces of Ireland / Demographics and politics	1	4	4
Toponymical list of counties of the United Kingdom / Northern ...	1	-	7
Múscraige / Notes	1	-	6

Individual Queries

We plot the difference between the LTR baseline and STR for the two query subsets in Figure 3.6. Table 3.9 lists the queries that we discuss below. The leftmost bar in Figure 3.6(a) corresponds to the query "*stocks*." For this broad query, there are two relevant and one highly relevant tables. LTR does not retrieve any highly relevant tables in the top 20, while STR manages to return one highly relevant table in the top 10. The rightmost bar in Figure 3.6(a) corresponds to the query "*ibanez guitars*." For this query, there are two relevant and one highly relevant tables. LTR produces an almost perfect ranking for this query, by returning the highly relevant table at the top rank, and the two relevant tables at ranks 2 and 4. STR returns a non-relevant table at the top rank, thereby pushing the relevant results down in the ranking by a single position, resulting in a decrease of 0.29 in NDCG@20.

The leftmost bar in Figure 3.6(b) corresponds to the query "*board games number of players*." For this query, there are only two relevant tables according to the ground truth. STR managed to place them in the 1st and 3rd rank positions, while LTR returned only one of them at position 13th. The rightmost bar in Figure 3.6(b) is the query "*cereals nutritional value*." Here, there is only one highly relevant result. LTR managed to place it in rank one, while it is ranked eighth by STR. Another interesting query is "*irish counties area*" (third bar from the left in Figure 3.6(b)), with three highly relevant and three relevant results according to the ground truth. LTR returned two highly relevant and one relevant results at ranks 1, 2, and 4. STR, on the other hand, placed the three highly relevant results in the top 3 positions and also returned the three relevant tables at positions 4, 6, and 7.

3.5 Summary and Conclusions

In this chapter, we have introduced and addressed the problem of ad hoc table retrieval: answering a keyword query with a ranked list of tables. Table retrieval is not interesting on its own, it is highly relevant to many table-related tasks like table completion (Chapter 5) and table generation (Chapter 6). We have developed a novel semantic matching framework, where queries and tables can be represented using semantic concepts (bag-of-entities and bag-of-categories) as well as continuous dense vectors (word and graph embeddings) in a uniform way. We have introduced multiple similarity measures for matching those semantic representations. For evaluation, we have used a purpose-built test collection based on Wikipedia tables. Finally, we have demonstrated substantial and significant improvements over a strong baseline.

Ad hoc table retrieval requires the user to issue a keyword query. When the user is already working on a table, we could retrieve related tables proactively based on that table as input, instead of resorting to a user-issued keyword query. Accordingly, we propose a separate table retrieval task, termed "query-by-table" in Chapter 4.

Chapter 4

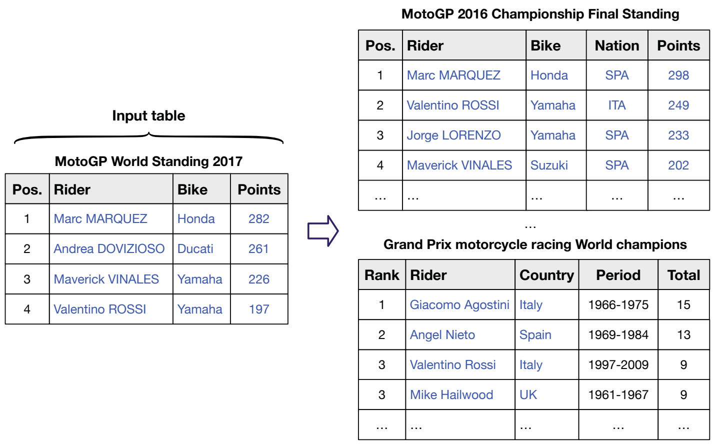
Query-by-Table

In Chapter 3, we have looked at how to rank tables using semantic matching in response to a keyword query. In a spreadsheet application, for instance, the table that a user is working on can in fact also be used as query. Therefore, we propose a novel table retrieval paradigm, referred to as *query-by-table*: given an input table, return a ranked list of relevant tables. At its core, this task boils down to computing the similarity between a pair of tables. We develop a theoretically sound framework for performing table matching. Our approach hinges on the idea of representing table elements in multiple semantic spaces, and then combining element-level similarities using a discriminative learning model, which is similar to that in Chapter 3. However, we further estimate the cross-element similarities for this task. Using a purpose-built test collection from Wikipedia tables, we demonstrate that the proposed approach delivers state-of-the-art performance.

This chapter is organized as follows. We introduce the query-by-table paradigm, adapt existing methods in Sect. 4.1, and present a discriminative approach that combines hand-crafted features from the literature in Sect. 4.2. We develop a general a table matching framework and specific instantiations of this framework in Sect. 4.3. We construct a purpose-built test collection in Sect. 4.4, perform a thorough experimental evaluation, and provide valuable insights and analysis in Sect. 4.5. Section 4.6 concludes this chapter.

4.1 Task Definition and Baseline Methods

In this section, we first introduce a novel table search paradigm, referred to as *query-by-table*: given an input table $\tilde{T}$, return a ranked list of related tables. That is, instead of requiring the user to express her information need by formulating a keyword query, we let her search for related tables by providing a table as input. This input table may be an incomplete table she currently works on or an existing table from earlier. Figure 4.1 illustrates the


Input table

MotoGP World Standing 2017

Pos.	Rider	Bike	Points
1	Marc MARQUEZ	Honda	282
2	Andrea DOVIZIOSO	Ducati	261
3	Maverick VINALES	Yamaha	226
4	Valentino ROSSI	Yamaha	197

MotoGP 2016 Championship Final Standing

Pos.	Rider	Bike	Nation	Points
1	Marc MARQUEZ	Honda	SPA	298
2	Valentino ROSSI	Yamaha	ITA	249
3	Jorge LORENZO	Yamaha	SPA	233
4	Maverick VINALES	Suzuki	SPA	202
...	...	...	...	...

...

Grand Prix motorcycle racing World champions

Rank	Rider	Country	Period	Total
1	Giacomo Agostini	Italy	1966-1975	15
2	Angel Nieto	Spain	1969-1984	13
3	Valentino Rossi	Italy	1997-2009	9
3	Mike Hailwood	UK	1961-1967	9
...	...	...	...	...

Figure 4.1: Query-by-table: given an input table, the system returns a ranked list of tables.

Table 4.1: Notation used in this chapter.

Symbol	Description
$\tilde{T}$	Input table
T	Candidate table
T_c	Table caption
T_p	Title of the page embedding the table
T_E	Set of entities in the table
T_H	Set of table column headings
T_D	Table data (or column data)

idea. Secondly, we present a number of existing methods from the literature that can be used to perform table matching. The objective is to compute the similarity between an input table $\tilde{T}$ and a candidate table T, expressed as $score(\tilde{T}, T)$. On the high level, all these methods operate by (i) subdividing tables into a number of *table elements* (such as caption, heading columns, and table data; Table. 4.1 lists all the notations), (ii) measuring the similarity between various elements of the input and candidate tables, and (iii) in case multiple elements are considered, combining these element-level similarities into a final score. Table 4.2 provides an overview of existing methods and the table elements they utilize.

Table 4.2: Table elements used in existing methods.

Method	T_c	T_p	T_E	T_H	T_D
Keyword-based search using T_E (Sect. 4.1.1)			✓		
Keyword-based search using T_H (Sect. 4.1.1)				✓	
Keyword-based search using T_c (Sect. 4.1.1)	✓				
Mannheim Search Join Engine (Sect.4.1.2)				✓	
Schema complement (Sect.4.1.3)			✓	✓	
Entity complement (Sect.4.1.4)			✓		
Nguyen et al. (Sect.4.1.5)				✓	✓
InfoGather (Sect.4.1.6)		✓		✓	✓

4.1.1 Keyword-based Search

Tables may be represented as unstructured documents, and ranked by means of keyword queries using standard document retrieval methods (e.g., BM25). Ahmadov et al. [2015a] use table entities and table headings as queries. Additionally, we also consider using the table caption as a query. Formally, we let d_T denote the document-based representation of table T. The candidate table's score is computed by taking the terms from $\tilde{T}_E$, $\tilde{T}_H$, or $\tilde{T}_c$ as the keyword query q: $score(\tilde{T}, T) = score_{BM25}(q, d_T)$.

4.1.2 Mannheim Search Join Engine

The Mannheim Search Join Engine (MSJE) [Lehmberg et al., 2015] provides table search functionality with the overall aim to extend an input table with additional attributes (i.e., columns). First, it uses exact column heading matching to filter tables that share at least one heading with the input table: $\mathcal{T} = \{T : |\tilde{T}_H \cap T_H| > 0\}$. Then, all tables in $\mathcal{T}$ are scored against the input table using the FastJoin matcher [Wang et al., 2011]. Specifically, Lehmberg et al. [2015] adapt edit distance with a threshold of δ to measure the similarity between the input and candidate tables' heading terms, $w(t_i, t_j)$, where $t_i \in \tilde{T}_H$ and $t_j \in T_H$. Terms in $\tilde{T}_H$ and T_H form a bipartite graph, with $w(t_i, t_j)$ as edge weights. Let $|\tilde{T}_H \tilde{\cap}_\delta T_H|$ denote the *maximum weighted bipartite matching score* on the graph's adjacency matrix, considering edges exceeding edit distance threshold. Then, the Jaccard similarity of two tables is expressed as:

$$score(\tilde{T}, T) = \frac{|\tilde{T}_H \tilde{\cap}_\delta T_H|}{|\{t : t \in \tilde{T}_H\}| + |\{t : t \in T_H\}| - |\tilde{T}_H \tilde{\cap}_\delta T_H|},$$

where $|\{t : t \in T_H\}|$ denotes the number of unique terms in the column headings of T.

4.1.3 Schema Complement

Das Sarma et al. [2012] search for related tables with the overall goal of extending the input table with additional attributes (referred to as *schema complement* in [Das Sarma et al., 2012]). For this task, they consider two factors: (i) the coverage of entities and (ii) the benefits of adding additional attributes. The final matching score is computed as:

$$score(\tilde{T}, T) = S_{EC}(\tilde{T}, T) \times S_{HB}(\tilde{T}, T). \tag{4.1}$$

The first component, entity coverage (EC), computes the entity overlap between two tables:

$$S_{EC}(\tilde{T}, T) = \frac{|\tilde{T}_E \cap T_E|}{|\tilde{T}_E|}. \tag{4.2}$$

The second component in Eq. (4.1) estimates the benefit of adding an additional column heading h to the input table:

$$HB(\tilde{T}_H, h) = \frac{1}{|\tilde{T}_H|} \sum_{\tilde{h} \in \tilde{T}_H} \frac{\#(\tilde{h}, h)}{\#(\tilde{h})} ,$$

where $\#(\tilde{h}, h)$ is number of tables containing both $\tilde{h}$ and h as column headings, and $\#(\tilde{h})$ is the number of tables containing $\tilde{h}$. The heading benefit between two tables, $S_{HB}(\tilde{T}, T)$, is computed by aggregating the benefits of adding all headings h from T to $\tilde{T}$:

$$S_{HB}(\tilde{T}, T) = aggr(HB(\tilde{T}_H, h)) .$$

The aggregation function $aggr()$ can be sum, average, or max.

4.1.4 Entity Complement

In addition to schema complement tables, Das Sarma et al. [2012] also search for *entity complement* tables, in order to augment the input table with additional entities (as rows). This method considers the relatedness between entities of the two tables:

$$score(\tilde{T}, T) = \frac{1}{|\tilde{T}_E||T_E|} \sum_{\tilde{e} \in \tilde{T}_E} \sum_{e \in T_E} sim(\tilde{e}, e),$$

where $sim(\tilde{e}, e)$ is a pairwise entity similarity measure. Specifically, we employ the *Wikipedia Link-based Measure* (WLM) [Milne and Witten, 2008], which estimates the semantic relatedness between two entities based on other entities they link to:

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

where $\mathcal{L}_e$ is the set of outgoing links of e (i.e., entities e links to) and $|\mathcal{E}|$ is the total number of entities in the knowledge base.

4.1.5 Nguyen et al.

Nguyen et al. [2015] match tables by considering both their headings and content (table data). These two similarities are combined using a linear mixture:

$$score(\tilde{T}, T) = \alpha \times sim_H(\tilde{T}, T) + (1 - \alpha) \times sim_D(\tilde{T}, T) \ .$$

The heading similarity sim_H is computed by first creating a similarity matrix between the heading terms of $\tilde{T}_H$ and T_H, as in Sect. 4.1.2. Next, an attribute correspondence subgraph $C \subseteq (|\tilde{T}_H| \times |T_H|)$ is obtained by solving the *maximum weighted bipartite sub-graph problem* [Anan and Avigdor, 2007]. Finally, heading similarity is computed as:

$$sim_H(\tilde{T}, T) = \frac{\sum_{(i,j) \in C} w_{t_i,t_j}(\tilde{T}_H, T_H)}{\max(|\tilde{T}_H|, |T_H|)} \ .$$

Data similarity is measured based on columns. Each table column is represented as a binary term vector, $\mathbf{T}_{D,i}$, where each element indicates the presence (1) or absence (0) of a given term in column i of table T. The similarity between two columns is measured by their cosine similarity. Table similarity considers all column combinations of $\tilde{T}$ and T. To account for the high number of possible combinations, for table each column, only the most similar column is considered from the other table:

$$sim_D(\tilde{T}, T) = \frac{1}{2}\big(\sum_i \max_j \cos(\tilde{\mathbf{T}}_{D,i}, \mathbf{T}_{D,j}) + \sum_j \max_i \cos(\tilde{\mathbf{T}}_{D,i}, \mathbf{T}_{D,j})\big) \ .$$

4.1.6 InfoGather

Following Yakout et al. [2012], we consider element-wise similarity across four table elements: table data, column values, page title, and column headings. Element-wise similarities are combined by training a linear regression scorer:

$$score(\tilde{T}, T) = \sum_x w_x \times sim_x(\tilde{T}, T) \ ,$$

where x is a given table element, $sim_x()$ is the element-level similarity score, and w_x is the weight (importance) of that element. Each table element is expressed as a term vector, denoted as $\tilde{\mathbf{T}}_x$ and $\mathbf{T}_x$ for element x of the input and candidate tables, respectively. Element-level similarity is then estimated using the cosine similarity between the two term vectors:

$$sim_x(\tilde{T}, T) = \cos(\tilde{\mathbf{T}}_x, \mathbf{T}_x) = \frac{\tilde{\mathbf{T}}_x \cdot \mathbf{T}_x}{||\tilde{\mathbf{T}}_x|| \times ||\mathbf{T}_x||} \ . \qquad (4.3)$$

Specifically, following [Yakout et al., 2012], for the table data and page title elements we use IDF weighting, while for column heading and column values, we employ TF-IDF weighting.

4.2 Using Hand-crafted Features

We combine the various table similarity measures from the previous section in a feature-based ranking framework. Additionally, we introduce a set of features to describe the input and candidate tables on their own. As we will show in our experimental section, this approach outperforms the best method from the literature by almost 30%. Therefore, even though the individual features are not regarded as novel, the rich feature set we introduce here does represent an important contribution.

4.2.1 Retrieval Framework

Formally, our goal is to learn a ranking model $h(\tilde{T}, T) = h(\vec{x}_{\tilde{T},T})$ that gives a real-valued score for an input and candidate table pair, or equivalently, to the corresponding feature vector $\vec{x}_{\tilde{T},T}$. The feature vector is defined as:

$$\begin{aligned}\vec{x}_{\tilde{T},T} =& \langle \varphi_1(\tilde{T}), \dots, \varphi_n(\tilde{T}), \\ & \varphi_{n+1}(T), \dots, \varphi_{2n}(T), \\ & \varphi_{2n+1}(\tilde{T}, T), \dots, \varphi_{2n+m}(\tilde{T}, T) \rangle \end{aligned} \tag{4.4}$$

There are two main groups of features. The first m features are used for representing the similarity between a pair of tables; these are described in Sect. 4.2.2. Then, $2n$ features are based on the characteristics of the input and candidate tables, respectively (n features each). These features are discussed in Sect. 4.2.3.

4.2.2 Table Similarity Features

We consider all element-level similarity scores from the individual methods in Sect. 4.1 as table similarity features. These are shown in Table 4.3, grouped by table elements.

4.2.3 Table Features

Additionally, we present a set of features that characterize individual tables. Table features are computed for both the input and candidate tables. They might be thought of as analogous to the query and document features, respectively, in document retrieval [Macdonald et al., 2012]. In fact, we adapt some features from document retrieval, such as query IDF score [Qin et al., 2010]. Specifically, we compute IDF for the table caption and page title elements, by summing up the term IDF scores: $IDF(f) = \sum_{t \in f} IDF(t)$. We further consider general table descriptors from [Bhagavatula et al., 2013], like the number of table rows, columns, and empty cells. Another group of features is concerned with the page in which the table is embedded. The includes page connectivity (inLinks and outLinks), page popularity (page

Table 4.3: Table similarity features. All values are in $[0, 1]$.

Element / Feature	**Source**
Page title ($\tilde{T}_p \leftrightarrow T_p$)	
InfoGather page title IDF similarity score (Sect. 4.1.6)	[Yakout et al., 2012]
Table headings ($\tilde{T}_H \leftrightarrow T_H$)	
MSJE heading matching score (Sect. 4.1.2)	[Lehmberg et al., 2015]
Schema complement schema benefit score (Sect. 4.1.3)	[Das Sarma et al., 2012]
InfoGather heading-to-heading similarity (Sect. 4.1.6)	[Yakout et al., 2012]
Nguyen et al. heading similarity (Sect. 4.1.5)	[Nguyen et al., 2015]
Table data ($\tilde{T}_D \leftrightarrow T_D$)	
InfoGather column-to-column similarity (Sect. 4.1.6)	[Yakout et al., 2012]
InfoGather table-to-table similarity (Sect. 4.1.6)	[Yakout et al., 2012]
Nguyen et al. table data similarity (Sect. 4.1.5)	[Nguyen et al., 2015]
Table entities ($\tilde{T}_E \leftrightarrow T_E$)	
Entity complement entity relatedness score (Sect. 4.1.4)	[Das Sarma et al., 2012]
Schema complement entity overlap score (Sect. 4.1.3)	[Das Sarma et al., 2012]

counts), and the table's importance within the page (tableImportance and tablePageFraction). Table 4.4 provides an overview of table features. Notice that these features largely overlap with those we considered in the previous chapter, cf. Table 3.2.

4.3 The CRAB Approach

This section presents our novel approach for table matching. Our contributions are twofold. We introduce a general element-oriented table matching framework in Sect. 4.3.1 followed by specific instantiations of this framework, referred to as CRAB, in Sect. 4.3.2.

4.3.1 Element-Level Table Matching Framework

We combine multiple table quality indicators and table similarity measures in a discriminative learning framework. Input and candidate table pairs are described as a feature vector, shown in Eq. (4.4). The main novelty lies in how table similarity is estimated. Instead of relying on hand-crafted features, like the ones presented in Sect. 4.2, we represent table elements in a uniform manner. Moreover, instead of relying of lexical matches, we perform the matching of table elements in multiple semantic spaces.

Let $\tilde{T}^y_{x1}$ denote element $x1$ of the input table $\tilde{T}$ in representation space y. Similarly, let T^y_{x2} denote element $x2$ of the candidate table T in representation space y. We then take table similarity features to be element-level matching

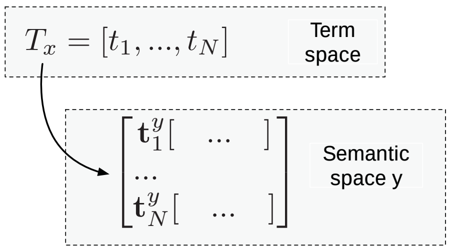


Figure 4.2: Representation of a table element T_x in the term and in a given semantic space y.

scores:

$$\varphi_i(\tilde{T}, T) = sim(\tilde{T}^y_{x1}, T^y_{x2}) \ ,$$

where $i \in [2n+1, 2n+m]$ and $sim()$ is a similarity function. Importantly, these similarity functions are applicable both to elements of the same type ($x1 = x2$), referred to as *element-wise matching* (e.g., caption vs. caption, headings vs. headings, etc.) and to elements of different types ($x1 \neq x2$), referred to as *cross-element matching* (e.g., caption vs. headings or headings vs. data). Next, we present various ways of representing table elements (Sect. 4.3.1), and measuring element-level similarity (Sect. 4.3.1).

Representing Table Elements

Each table element, T_x, is represented both in a *term space* and in a *semantic space*. We start with the former one. T_x is described as a weighted vector of terms, where terms may be words or entities. Formally, $T_{\vec{x}} = [t_1, \ldots, t_N]$, where t_i corresponds to the weight of the ith term in the vocabulary. For words, the vocabulary is the set of unique words in the table corpus, for entities it is the set of entries in a knowledge base. We also represent each table element in a semantic space. Given a semantic space y, each term t_i is described by a corresponding *embedding vector*, $\vec{t}^y_i$. The space of embeddings may be words, entities, or graphs (cf. Sect. 4.3.2).

In summary, each table element is represented in the term space by a term vector $T_{\vec{x}}$, and each term $t_i \in T_{\vec{x}}$ is represented by a semantic vector $\vec{t}^y_i$. Note that the term space serves only as an intermediate representation, to help map table elements to semantic space y. The subsequent element-level matching will only be performed in this semantic space. See Fig. 4.2 for an illustration.

Measuring Element-level Similarity

We estimate the similarity between two table elements, $\tilde{T}_{x_1}$ and T_{x_2}, based on their semantic representations. Notice that these semantic representations

Table 4.4: Table features.

Feature	Description	Source
#rows	Number of rows in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
#cols	Number of columns in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
#of NULLs	Number of empty table cells	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
IDF(T_c)	Table caption IDF	[Qin et al., 2010]
IDF(T_p)	Table page title IDF	[Qin et al., 2010]
inLinks	Number of in-links to the page embedding the table	[Bhagavatula et al., 2013]
outLinks	Number of out-links from the page embedding the table	[Bhagavatula et al., 2013]
pageViews	Number of page views	[Bhagavatula et al., 2013]
tableImportance	Inverse of number of tables on the page	[Bhagavatula et al., 2013]
tablePageFraction	Ratio of table size to page size	[Bhagavatula et al., 2013]

(that is, the embedding vectors $\vec{t}_i^y$) are on the term level and not on the element level. Thus, the term embedding vectors need to be aggregated on the element level. Following in spirit with the approach taken in Chapter 3, we present four specific element-level similarity methods. These are roughly analogous to the early and late fusion strategies in [Snoek et al., 2005, Zhang and Balog, 2017b]. We refer to Fig. 4.3 for an illustration.

One strategy, referred to as *early fusion*, represents each table element T_x in semantic space y by combining the term-level semantic vectors to a single element-level semantic vector, $\vec{C}_x^y$. We take the weighted centroid of term-level semantic vectors:

$$\vec{C}_x^y[i] = \frac{\sum_{j=1}^{N} t_j \times \vec{t}_j^y[i]}{\sum_{j=1}^{N} t_j} ,$$

where $[i]$ refers to the ith element of the vector. Then, the similarity of two table elements is taken to be the cosine similarity of their respective centroid vectors:

$$sim_{early}(\tilde{T}_{x_1}, T_{x_2}) = \cos(\vec{C}_{x1}^y, \vec{C}_{x2}^y) .$$

According to another strategy, referred to as *late fusion*, we first compute the cosine similarities between all pairs of semantic vectors. Then, these term-level similarity scores are aggregated into an element-level score:

$$sim_{late}(\tilde{T}_{x_1}, T_{x_2}) = aggr(\{\cos(\vec{t}_1, \vec{t}_2) : \vec{t}_1 \in \tilde{T}_{\vec{x}_1}^y, \vec{t}_2 \in T_{\vec{x}_2}^y\}) ,$$

where $aggr()$ is an aggregation function. Specifically, we use $max()$, $sum()$, and $avg()$ as aggregation functions.

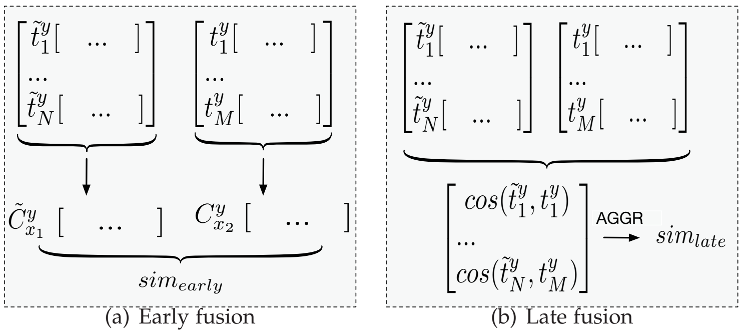


Figure 4.3: Illustration of element-level similarity methods.

4.3.2 CRAB

We detail a specific instantiation of our framework, which includes the representation of table elements (Sect. 4.3.2) and the element-level similarity scores that are used as ranking features (Sect. 4.3.2).

Representing Table Elements

We split tables into the following elements (see Fig. 2.2 for an illustration) and represent them in (at most) two term spaces, words and entities, as follows:

- **Table headings** (T_H) Table headings are represented only as words, since entity occurrences in headings are extremely rare. In the case that entities appear in headings, we assign them to the table data element.
- **Table data** (T_D) The contents of table cells are used both as words and as entities. For the latter, entity mentions need to be recognized and disambiguated; such annotations be made readily available as markup (e.g., in Wikipedia tables) or may be obtained automatically using entity linking techniques [Shen et al., 2015].
- **Table topic** (T_t) For simplicity, we combine table caption and page title into a single *table topic* element. We can directly use this text for representing the table topic in the word space. To obtain entities for the table topic, we use the table topic text as a search query to retrieve the top-k relevant entities from a knowledge base. Specifically, we employ the MLM [Hasibi et al., 2017a] retrieval method with $k = 10$.
- **Table entities** (T_E) Many relational tables have a core entity column [Bhagavatula et al., 2015, Venetis et al., 2011], while the rest of columns represent attributes of those entities. For this table element we only keep entities from the table's *core column*, i.e., the column with the

highest entity rate. We estimate entity rate by calculating the number of column cells that contain an entity and divide it by the number of rows.

We consider three semantic spaces for representing table elements: word, entity, and graph embeddings. These are explained below.

Word embeddings Each table element is represented in the term space as a TF-IDF-weighted vector of words. I.e., $t_i \in T_x$ refers to the TF-IDF weight of the ith word in the vocabulary. Then, each word is represented in the semantic space $y = w$ by a word embedding vector $\mathbf{t}_i^w$. Specifically, we use pre-trained Word2vec [Mikolov et al., 2013, Pennington et al., 2014] vectors.

Graph embeddings Each table element is represented in the term space as a binary vector of entities. I.e., $t_i \in T_x$ is 1 if the ith entity in the knowledge base appears in table element T_x, and is 0 otherwise. Then, each entity is represented in the semantic space $y = g$ by a graph embedding vector $\mathbf{t}_i^g$. Specifically, we use pre-trained Graph2vec [Ristoski and Paulheim, 2016] vectors.

Entity embeddings We use the same term space representation as for graph embedding, i.e., each table element is described as a binary vector of entities. Then, each entity t_i is represented in the semantic space $y = e$ as a vector of linked entities. I.e., the dimensionality of $\mathbf{t}_i^e$ is the total number of entities in the knowledge base. The jth element of the related entity vector is expressed as $\mathbf{t}_i^g[j] = \mathbb{1}(e_j)$, where $\mathbb{1}$ is a binary indicator function that returns 1 if e_i and e_j link to each other, otherwise returns 0.

Table Similarity Features

Existing methods have only considered matching between elements of the same type, referred to as *element-wise* matching. Our framework also enables us to measure the similarities between elements of different types in a principled way, referred to as *cross-element* matching. Finally, as before, we can also utilize table features that characterize the input and candidate tables. Below, we detail the set of features used for measuring element-level similarity.

Element-wise similarity We compute the similarity between elements of the same type from the input and candidate tables. Each table element may be represented in up to three semantic spaces. Then, in each of those spaces, similarity is measured using the four element-level similarity measures (early, late-max, late-sum, and late-avg). Element-wise features are summarized in the left half of Table 4.5.

Table 4.5: Element-wise and cross-element features used in CRAB. The dimension is $r \times \#s \times \#m$, where r is reflection (1 for element-wise and 2 for cross-element), s is the number of semantic spaces, and m is the number of element-wise similarity measures.

Element	**Dimension**	**Element**	**Dimension**
$\tilde{T}_H$ to T_H	$1 \times 1 \times 4 = 4$	$\tilde{T}_H$ to T_t	$2 \times 1 \times 4 = 8$
$\tilde{T}_D$ to T_D	$1 \times 3 \times 4 = 12$	$\tilde{T}_H$ to T_D	$2 \times 1 \times 4 = 8$
$\tilde{T}_E$ to T_E	$1 \times 2 \times 4 = 8$	$\tilde{T}_D$ to T_t	$2 \times 3 \times 4 = 24$
$\tilde{T}_t$ to T_t	$1 \times 3 \times 4 = 12$	$\tilde{T}_D$ to T_E	$2 \times 2 \times 4 = 16$
		$\tilde{T}_t$ to T_E	$2 \times 2 \times 4 = 16$
Total	36		72

Cross-element similarity This approach compares table elements of different types in an asymmetrical way. Each pair of elements need to be represented in the same semantic space. Then, the same element-level similarity measures may be applied, as before. We list the cross-element similarity features in the right half of Table 4.5.

We present four specific instantiations of our table matching framework, by considering various combinations of the three main groups of features. These instantiations are labelled as CRAB-1 .. CRAB-4 and are summarized in Table 4.6.

4.4 Test collection

We introduce our test collection, which consists of a table corpus, a set of query tables, and corresponding relevance assessments.

4.4.1 Table Corpus

We use the WikiTables corpus [Bhagavatula et al., 2015], which contains 1.6M tables extracted from Wikipedia. The knowledge base we use is DBpedia (version 2015-10). We restrict ourselves to entities which have a short textual summary (abstract) in the knowledge base (4.6M in total). Tables are preprocessed as follows. For each cell that contains a hyperlink we check if it points to an entity that is present in DBpedia. If yes, we use the DBpedia identifier of the linked entity as the cell's content (with redirects resolved); otherwise, we replace the link with the anchor text (i.e., treat it as a string).

Table 4.6: Features used in various instantiations of our element-wise table matching framework. Element-wise and cross-element features are summarized in Table 4.5, table feature are listed in Table 4.4.

Method	**Table similarity features**		
	Element-wise	**Cross-element**	**Table features**
CRAB-1	√		
CRAB-2	√		√
CRAB-3		√	√
CRAB-4	√	√	√

4.4.2 Query Tables and Relevance Assessments

We sample 50 Wikipedia tables from the table corpus to be used as queries. Each table is required to have at least five rows and three columns. These tables cover a diverse set of topics, including sports, music, films, food, celebrities, geography, and politics.

Ground truth relevance labels are obtained as follows. For each input table, three keyword queries are constructed: (i) caption, (ii) table entities (entities from table plus the entity corresponding to the Wikipedia page in which the table is embedded), and (iii) table headings. Each keyword query is used to retrieve the top 150 results, resulting in at most 450 candidate tables for each query table. All methods that are compared in the experimental section operate by reranking these candidate sets. For each method, the top 10 results are manually annotated.

Each query-table pair is judged on a three point scale: non-relevant (0), relevant (1), and highly relevant (2). A table is highly relevant if it is about the same topic as the input table, but contains additional novel content that is not present in the input table. A table is relevant if it is on-topic, but it contains limited novel content; i.e., the content largely overlaps with that of the input table. Otherwise, the table is not relevant; this also includes tables without substantial content. Three colleagues were employed and trained as annotators. We take the majority vote as the relevance label; if no majority vote is achieved, the mean score is used as the final label. To measure inter-annotator agreement, we compute the Fleiss Kappa test statistics, which is 0.6703. According to Fleiss et al. [1971], this is considered as substantial agreement.

4.4.3 Evaluation Metrics

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

4.5 Evaluation

In this section, we report on the experiments we conducted to answer our research questions.

4.5.1 Experimental Setup

The experimental configurations of the various methods are as follows. For *keyword-based search* (Sect. 4.1.1), the T_E and T_H methods query an index of the table corpus against the respective fields, while the T_c variant searches against both the caption and catchall fields; all the three methods use BM25. For the *Mannheim Search Join Engine* (Sect. 4.1.2), the edit distance threshold is set to $\delta = 0.8$. For *schema complement* (Sect. 4.1.3), the heading frequency statistics is calculated based on the Wikipedia table corpora and the heading similarity is aggregated using average. For *entity complement* (Sect. 4.1.4), WLM is based on entity out-links. The data similarity threshold is set the same as for string comparison, i.e., $\delta = 0.8$. To parse terms in attribute values, we remove stopwords and HTML markup, and lowercase tokens. For *Nguyen et al.* (Sect. 4.1.5), the smoothing parameter value is taken from [Nguyen et al., 2015] to be $\alpha = 0.5$. *InfoGather* (Sect. 4.1.6) is trained using linear regression with coordinate ascent. All methods introduced by us, i.e., *HCF-X* and *CRAB-X*, are trained using Random Forest Regression with 5-fold cross-validation; the number of trees is 1000 and the maximum number of features is 3.

4.5.2 Baselines

Table 4.7 presents the evaluation results for the eight existing methods from the literature, which we presented in Sect. 4.1. Among the three keyword-based search methods, which operate on a single table element (top 3 lines), the one that uses table headings as the keyword query performs the best, followed by table entities and table caption. The methods in lines 4–8 consider multiple table elements; all of these outperform the best single-element method. The approach that performs best among all, by a large margin, is InfoGather, which incorporates four different table elements (cf. Sect. 4.1.6). Consequently, we will test our methods against InfoGather.

4.5.3 Results

Table 4.8 compares the evaluation results of the methods we developed in this paper against InfoGather. HCF-1, which combines all table similarity

Table 4.7: Evaluation results for existing methods from the literature. Best scores for each metric are boldfaced.

Method	**NDCG@5**	**NDCG@10**
Keyword-based search using T_E (Sect. 4.1.1)	0.2001	0.1998
Keyword-based search using T_H (Sect. 4.1.1)	0.2318	0.2527
Keyword-based search using T_c (Sect. 4.1.1)	0.1369	0.1419
Mannheim Search Join Engine (Sect. 4.1.2)	0.3298	0.3131
Schema complement (Sect. 4.1.3)	0.3389	0.3418
Entity complement (Sect. 4.1.4)	0.2986	0.3093
Nguyen et al. (Sect. 4.1.5)	0.2875	0.3007
InfoGather (Sect. 4.1.6)	**0.4530**	**0.4686**

features from existing approaches, achieves 18.27% improvement upon infoGather in terms of NDCG@10, albeit the differences are not statistically significant. HCF-2 incorporates additional table features, which leads to substantial (29.11% for NDCG@10) and significant improvements over InfoGather. The bottom block of Table 4.8 presents the evaluation results for four specific instantiations of our table matching framework (cf. Table 4.6). Recall that CRAB-1 employs only table similarity features, thus it is to be compared against HCF-1. CRAB-2..4 additionally consider table features, which corresponds to the settings in HCF-2. We find that CRAB-1 and CRAB-2 outperform the respective HCF method, while CRAB-4 is on par with it. None of the differences between CRAB-X and the respective HCF method are statistically significant. The best overall performer is CRAB-2, with a relative improvement of 36.2% for NDCG@5 and 33.7% for NDCG@10 over InfoGather. Figure 4.4 shows performance differences on the level of individual input tables between InfoGather and CRAB-2. Clearly, several tables are improved by a large margin, while only a handful of tables are affected negatively.

The summary of our findings thus far is that our semantic table element representation with element-wise matching is very effective. We can achieve the same performance as a state-of-the-art approach that relies on handcrafted features (CRAB-1 vs. HCF-1 and CRAB-2 vs. HCF-2). With that, we have accomplished our main research objective. We further observe that cross-element matching is less effective than element-wise matching (CRAB-3 vs. CRAB-2). Combining all element-wise and cross-element features performs worse than using only the former (CRAB-4 vs. CRAB-2).

Now that we have assessed the overall effectiveness of our approach, let us turn to answering a series of more specific research questions.

RQ2/A Which of the semantic representations (word-based, graph-based, or

Table 4.8: Table retrieval evaluation results of our methods against the best existing method. Significance is tested against Infogather. Highest scores are in boldface.

Method	NDCG@5	NDCG@10
InfoGather	0.4530	0.4686
HCF-1 (feats. from Table 4.3)	0.5382	0.5542
HCF-2 (feats. from Tables 4.3 and 4.4)	**0.5895†**	**0.6050†**
CRAB-1	0.5578	0.5672
CRAB-2	**0.6172‡**	**0.6267‡**
CRAB-3	0.5140	0.5282
CRAB-4	0.5804†	0.6027†

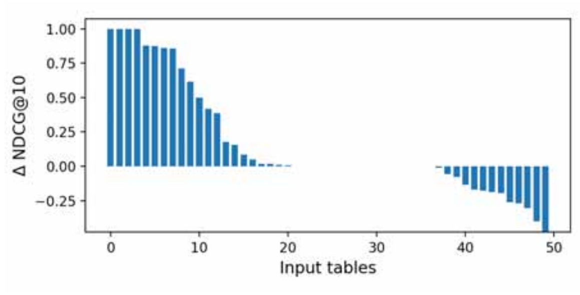


Figure 4.4: Performance difference between InfoGather (baseline) and CRAB-2 on the level of individual input tables. Positive bars indicate the advantage of CRAB-2.

Table 4.9: Element-wise similarities for various semantic representations. Rows and columns corresponds to elements of the input and candidate tables, respectively. The evaluation metric is NDCG@10. The best scores for each row are in boldface.

	Word-based				Graph-based				Entity-based		
	T_t	T_H	T_D		T_t	T_E	T_D		T_t	T_E	T_D
$\tilde{T}_t$	**0.2814**	0.0261	0.0436	$\tilde{T}_t$	**0.2765**	0.0546	0.0430	$\tilde{T}_t$	**0.4796**	0.0808	0.0644
$\tilde{T}_H$	0.0336	**0.1694**	0.0288	$\tilde{T}_E$	**0.0700**	0.0679	0.0501	$\tilde{T}_E$	0.0705	0.0617	**0.0725**
$\tilde{T}_D$	0.0509	0.0183	**0.1276**	$\tilde{T}_D$	**0.1012**	0.0423	0.0259	$\tilde{T}_D$	**0.1052**	0.0812	0.0610

entity-based) is the most effective for modeling table elements?

Table 4.10 displays results for each of the three semantic representations. Among those, entity-based performs the best, followed by word-based and graph-based. The differences between entity-based and word-based are significant ($p < 0.01$), but not between the other pairs of representations. Inter-

Table 4.10: Comparison of semantic representations.

Semantic Repr.	**NDCG@5**	**NDCG@10**
Word-based	0.3779	0.3906
Graph-based	0.3012	0.3376
Entity-based	0.4484	0.4884
Combined	0.5578	0.5672

estingly, the entity-based representation delivers performance that is comparable to that of the best existing method, InfoGather (cf. Table 4.7). When combing all three semantic representations (line 4, which is the the same as CRAB-1 in Table 4.8), we obtain substantial and significant improvements (p <0.01) over each individual representation. This shows the complimentary nature of these semantic representations.

RQ2/B Which of the two element-level matching strategies performs better, element-wise or cross-element?

We found that adding all the cross-element similarities hurts (CRAB-4 vs. CRAB-2 in Table 4.8). In order to get a better understanding of how the element-wise and cross-element matching strategies compare against each other, we break down retrieval performance for all table element pairs according to the different semantic representations in Table 4.9. That is, we rank tables by measuring similarity only between that pair of elements (4 table similarity features in total). Here, diagonal cells correspond to element-wise matching and all other cells correspond to cross-element matching. We observe that element-wise matching works best across the board. This is in line with our earlier findings, i.e., CRAB-2 vs. CRAB-3 in Table 4.8. However, for graph-based and entity-based representations, there are several cases where cross-element matching yields higher scores than element-wise matching. Notably, input table data ($\tilde{T}_D$) has much higher similarity against the topic of the candidate table (T_t) than against its data (T_D) element, for both graph-based and entity-based representations. This shows that cross-element matching does have merit for certain table element pairs. We perform further analysis in Sect. 4.5.4.

RQ2/C How much do different table elements contribute to retrieval performance?

To explore the importance of table elements, we turn to Table 4.9 once again. We first compare the results for element-wise similarity (i.e., the diagonals) and find that among the four table elements, table topic ($\tilde{T}_t \leftrightarrow T_t$) contributes the most and table data ($\tilde{T}_D \leftrightarrow T_D$) contributes the least. Second, our observations for cross-element matching are as follows. Using word-

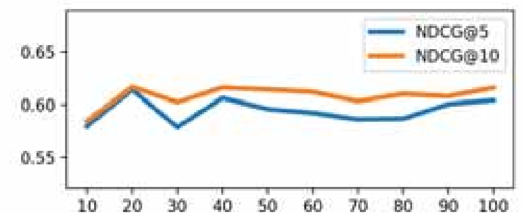


Figure 4.5: Performance in terms of NDCG with different number of top features utilized.

based representation, table data ($\tilde{T}_D$) is the most important element for the input table, while for the candidate table it is table topic (T_t). Interestingly, for graph-based and entity-based representations it is exactly the other way around: the most important input table element is topic ($\tilde{T}_t$), while the most important candidate table element is data (T_D).

4.5.4 Further Analysis

Now that we have represented our experimental results, we perform further performance analysis on individual features and on input table size.

Feature Analysis

To understand the contributions of individual features, we first rank all features based on Gini importance. Then, we incrementally add features in batches of 10, and plot the corresponding retrieval performance in Figure 4.5. We observe that we can reach peak performance with using only the top-20 features. Let us take a closer look at those top-20 features in Figure 4.6. We use color coding to help distinguish between the three main types of features: element-wise, cross-element, and table features. Then, based on these feature importance scores, we revisit our research questions. Concerning semantic representations (**RQ2/A**), there are 8 word embedding, 7 entity embedding, and 3 graph embedding features in the top 20. Even though there are slightly more features using word embedding than entity embeddings, the latter features are much higher ranked (cf. Fig. 4.6). Thus, the entity-based semantic representation is the most effective one. Comparing matching strategies (**RQ2/B**), the numbers of element-wise and cross-wise features are 15 and 3, respectively. This indicates a substantial advantage of element-wise strategies. Nevertheless, it shows that incorporating the similarity between elements of different types can also be beneficial. Additionally, there are 2 table features in the top 20. As for the importance of table elements (**RQ2/C**), table topic (T_t) is clearly the most important one; 8 out

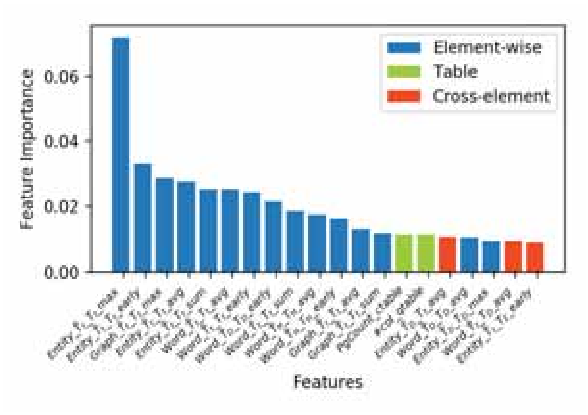


Figure 4.6: Top-20 features in terms of Gini importance.

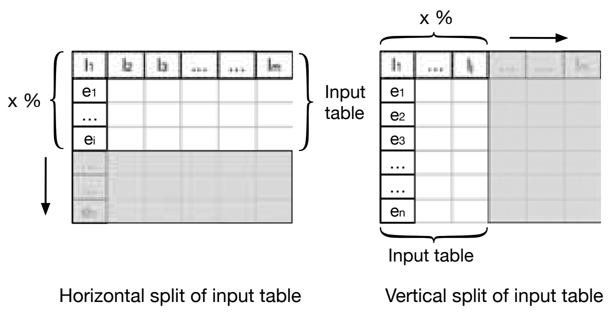


Figure 4.7: Performance analysis using only a portion of the table as input.

of the top 10 features consider that element. In summary, our observations based on the top-20 features are consistent with our earlier findings.

Input table size

Next, we explore how the size of the input table affects retrieval performance. Specifically, we vary the input table size by splitting it horizontally (varying the number of rows) or vertically (varying the number of columns), and using only a portion of the table as input; see Fig. 4.7 for an illustration. We explore four settings by setting the split rate x between 25% and 100% in steps of 25%. Figure 4.8 plots retrieval performance against input table size.

We observe that growing the table, either horizontally or vertically, results in proportional increase in retrieval performance. This is not surprising, given that larger tables contain more information. Nevertheless, being able to utilize this extra information effectively is an essential characteristic of our table matching framework.

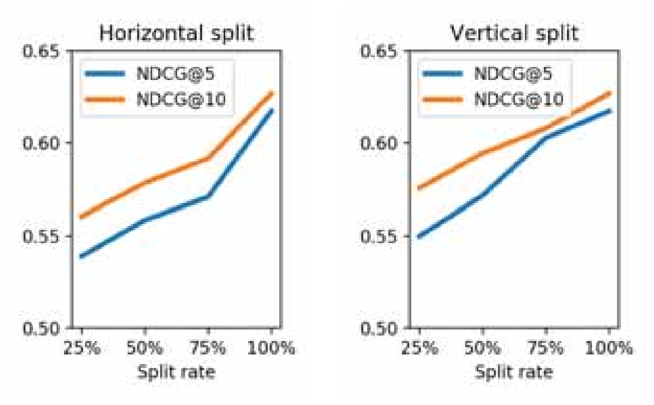


Figure 4.8: Performance of CRAB-2 with respect to (relative) input table size, by varying the number of rows (Left) or columns (Right).

4.6 Summary and Conclusions

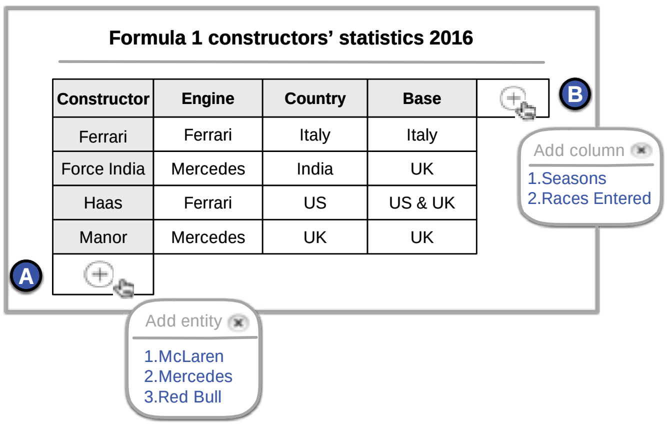
In this chapter, we have introduced and addressed the *query-by-table* task: returning a ranked list of tables that are related to a given input table. We have proposed a novel element-oriented table matching framework that represents table elements uniformly and considers their similarity in multiple semantic spaces. This framework can incorporate the similarity between table elements that are of the same type (element-wise matching) as well as those that are of different types (cross-element matching). We have further presented four specific instantiations of this general framework and considered word-based, graph-based, and entity-based semantic representations. For evaluation, we have developed a standard test collection based on Wikipedia tables, and demonstrated that our approach delivers state-of-the-art performance.

Table retrieval can be utilized in a range of table-centric tasks by providing related tables with additional tabular data. In the next chapter, we will illustrate how we can assist users in completing tables by leveraging related tables and knowledge bases.

Chapter 5

Table Completion

Our motivation for this chapter is to assist users with completing tables by providing smart assistance capabilities. The scenario we consider is the following. We assume a user, working with a table, at some intermediate stage in the process. At this point, she has already set the caption of the table and entered some data into the table. The table is assumed to have a column header (located above the first content row), which identifies each column with a unique label. Recall that in this thesis we concentrate on one particular family of tables, namely, relational tables (tables with an entity focus). Our objective is to aid the user by offering "smart suggestions," that is, recommending (i) additional entities (rows) and (ii) additional column headings, to be added to the table. Accordingly, we introduce and focus on two specific tasks: populating rows with additional instances (entities) and populating columns with new headings. See Figure 5.1 for an illustration. The task of row population relates to the task of *entity set expansion* [Das Sarma et al., 2012, Bron et al., 2013, Metzger et al., 2014, He and Xin, 2011, Wang and Cohen, 2008, Wang et al., 2015a], where a given set of seed entities (examples) is to be completed with additional entities. Here, we also have a seed set of entities from the leftmost table column. But, in addition to that, we can also make use of the column heading labels and the caption of the table. We show in our experiments, that utilizing these can lead to substantial improvements over using only the seed entities. The second task, column population, shares similarities with the problem of *schema complement* [Das Sarma et al., 2012, Lehmberg et al., 2015, Bhagavatula et al., 2013, Yakout et al., 2012], where a seed table is to be complemented with related tables that can provide additional columns. Many of these existing approaches utilize the full table content and also address the task of merging data into the seed table. Here, our focus is only on finding proper column headings, using the same sources as for row population (i.e., leftmost column, header row, and table caption). We develop generative probabilistic



Formula 1 constructors' statistics 2016

Constructor	Engine	Country	Base
Ferrari	Ferrari	Italy	Italy
Force India	Mercedes	India	UK
Haas	Ferrari	US	US & UK
Manor	Mercedes	UK	UK

Figure 5.1: Envisioned user interface. Column headings and the leftmost column are marked with a grey background. The user can populate the table with (A) additional entities (rows) and (B) additional column headings. The suggestions in the pop-ups are updated dynamically as the content of the table changes.

models for both tasks. For estimating the components of these models, we consider a knowledge base as well as a large table corpus. Our experimental evaluation simulates the various stages of the user entering content into an actual table. A detailed analysis of the results shows that the models' components are complimentary and that our methods outperform existing approaches from the literature.

The chapter is organized as follows. In Sect 5.1 we introduce and formalize two specific tasks for providing intelligent assistance with tables: row population and column population. We present generative probabilistic methods for both tasks, which combine existing approaches from the literature with novel components in Sect. 5.2 and Sect. 5.3. We design evaluation methodology and develop a process that simulates a user through the process of populating a table with data in Sect. 5.4. We perform an experimental evaluation and carry out a detailed analysis of performance in Sect. 5.5 and Sect. 5.6. Section 5.7 concludes this chapter.

5.1 Problem Statement

In this section, we provide a formal description of the tasks we propose to undertake. We refer to Table 5.1 for the notation used in this chapter.

Definition 5.1 (Table) *A table T is grid of cells, which hold values, arranged in $n+1$ rows and m columns. The top row is a special designated place, where the*

Table 5.1: Notation used in this chapter.

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

column headings reside. It is followed by n regular (content) rows. We let $L = (l_1, \ldots, l_m)$ be the list of column heading labels. In addition to the grid content, the table also has a caption c.

Definition 5.2 (Entity-Focused Table) *A table is said to be entity-focused, if its leftmost column contains only entities as values, and those entities are unique within the column (a particular relational table with a fixed core column). We let $E = (e_1, \ldots, e_n)$ be the list of entities corresponding to the leftmost table column. I.e., the table takes the following shape:*

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

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

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

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

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

Definition 5.3 (Row Population) *Row population is the task of generating a ranked list of entities to be added to the leftmost column of a given seed table, as* e_{n+1}.

Definition 5.4 (Column Population) *Column population is the task of generating a ranked list of column labels to be added to the column headings of a given seed table, as* l_{m+1}.

In the following two sections, we present our approaches for row and column population. Following prior studies [Lehmberg et al., 2015, Bhagavatula et al., 2013, Yakout et al., 2012, Ahmadov et al., 2015a, Das Sarma et al., 2012, Yin et al., 2016, Cafarella et al., 2008a, Venetis et al., 2011, Crestan and Pantel, 2011, Sekhavat et al., 2014a, Wang et al., 2012, 2015a, Bhagavatula et al., 2015], we rely heavily on the availability of a large table corpus as an external resource (which, in our case, is extracted from Wikipedia). Additionally, we also exploit information stored about entities in a knowledge base (in our case, DBpedia). Further specifics about our data sources are provided in Sect. 5.4.1.

5.2 Populating Rows

In this section, we address problem of row population using a two-step approach. We assume that a seed table is given, with a list of n seed entities E, a list of m seed column labels L, and a table caption c. The task is to generate a ranked list of suggestions for entity e_{n+1}, which may be added to the seed table as a new row. First, we identify a set of candidate entities (Sect. 5.2.1), and then rank them in a subsequent entity ranking step (Sect. 5.2.2).

5.2.1 Candidate Selection

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

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

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

5.2.2 Ranking Entities

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

$$\begin{aligned} P(e|E,L,c) &= \frac{P(E,L,c|e)P(e)}{P(E,L,c)} \\ &= \frac{P(E|e)P(L|e)P(c|e)P(e)}{P(E)P(L)P(c)} \\ &\propto P(e|E)P(L|e)P(c|e) \ . \end{aligned} \tag{5.1}$$

In the last step, we rewrote $P(E|e)$ using Bayes' rule (which cancelled out $P(e)$ and $P(E)$). We further dropped the probabilities $P(L)$ and $P(c)$ from the denominator, since those are the same across all candidate entities and thus do not influence their ranking. Then, entities are ranked by multiplying (i) the posteriori probability $P(e|E)$ that expresses entity similarity, (ii) the column labels likelihood $P(L|e)$, and (iii) the caption likelihood $P(c|e)$. The reason for keeping the latter two probabilities conditioned on the candidate entity is that column labels and captions are very short. In those cases, the candidate entity offers a richer observation. Below, we discuss the estimation of each of these probabilities.

Note that entities may be ranked using any subset of the components in Eq. (5.1). We explore all possible combinations in our experimental section (Sect. 5.5). It is our expectation that using all three sources of evidence (seed entities, seed column labels, and table caption) would result in the best performance.

5.2.3 Entity Similarity

The estimation of $P(e|E)$ corresponds to the task of *entity list completion* (also known as *set/concept expansion* or *query by example*): given a small set of seed entities, complement this set with additional entities. The general idea is to measure the semantic similarity between the candidate entity and the set of seed entities. One line of prior work [Bron et al., 2013, Metzger et al., 2014] relies on a knowledge base for establishing this semantic similarity. Another family of approaches [Das Sarma et al., 2012, Wang and Cohen, 2008, Wang et al., 2015a] leverages a large table corpus for collecting co-occurrence statistics. We combine both these sources in a single model:

$$P(e|E) = \lambda_E P_{KB}(e|E) + (1-\lambda_E) P_{TC}(e|E) \;, \tag{5.2}$$

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

Estimation Using a Knowledge Base

Bron et al. [2013] create a structured entity representation for each entity from the RDF triples describing that entity. The structured representation of an entity is comprised by the set of relations of the entity. Each relation r is modeled as a pair, by removing the entity itself from the triples. E.g., given the triple ⟨`dbr:Japan, dbo:capital, dbr:Tokyo`⟩ describing the entity Japan, the corresponding relation becomes (`dbo:capital, dbr:Tokyo`). We write $\hat{e}$ to denote the structured representation of entity e. Formally, given a set of subject-predicate-object (s, p, o) triples describing the entity (i.e., the entity stands either as subject or object):

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

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

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

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

Instead of using a single model built for the set of seed entities, we also explore an alternative approach by taking the average pairwise similarity between the candidate and seed entities (similar in spirit to He and Xin [2011] and Das Sarma et al. [2012]):

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

where $\mathrm{sim}(e, e_i)$ is a similarity function. We consider two alternatives for this function. The first is the *Wikipedia Link-based Measure* (WLM) [Milne and

Witten, 2008], which estimates the semantic relatedness between two entities based on other entities they link to:

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

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

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

Estimation Using a Table Corpus

Another way of establishing the similarity between a candidate entity e and the set of seed entities E is to obtain co-occurrence statistics from a table corpus (as in [Das Sarma et al., 2012] and [Ahmadov et al., 2015a]). We employ a maximum likelihood estimator:

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

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

5.2.4 Column Labels Likelihood

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

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

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

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

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

The exact label matching probability is estimated using:

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

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

5.2.5 Caption Likelihood

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

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

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

$$P_{KB}(t|\theta_e) = \frac{tf(t,e) + \mu P(t|\theta)}{|e| + \mu} , \tag{5.3}$$

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

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

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

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

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

5.3 Populating Columns

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

5.3.1 Candidate Selection

We use (i) the table caption, (ii) table entities, and (iii) seed column heading labels to search for similar tables. The searching method is the same as in Sect. 5.2.1, i.e., we use BM25 similarity using either of (i)–(iii) to get a ranking of tables from the table corpus. From these tables, we extract the column heading labels as candidates (excluding the seed column labels). When searching is done using the seed column labels as query, our method is equivalent to the FastJoin matcher [Wang et al., 2014] (which was also adopted in [Lehmberg et al., 2015]).

5.3.2 Ranking Column Labels

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

Baseline Approach

Das Sarma et al. [2012] consider the "benefits" of additional columns. The benefit of adding l to table T is estimated as follows:

$$P(l|L) = LB(L, l) = \frac{1}{|L|} \sum_{l_1 \in L} cs(l_1, l) \ , \tag{5.4}$$

where L denotes column labels and cs is the AcsDB (*Attribute Correlation Statistics Database*) schema frequency statistics [Cafarella et al., 2008a], which is given in Eq. (5.5). It is more effective to derive the benefit measure by considering the co-occurrence of pairs of labels, rather than the entire set of labels [Das Sarma et al., 2012]. Eq. (5.5) determines the consistency of adding a new label l to an existing label l_1:

$$cs(l_1, l) = P(l|l_1) = \frac{\#(l_1, l)}{\#(l_1)} \ , \tag{5.5}$$

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

Our Approach

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

By applying the law of total probability, we get:

$$P(l|E,c,L) = \sum_{T} P(l|T)P(T|E,c,L) \ ,$$

where $P(l|T)$ is the label's likelihood given a candidate table (see Sect. 5.3.3), and $P(T|E,c,L)$ expresses that table's relevance (see Sect. 5.3.4).

5.3.3 Label Likelihood

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

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

5.3.4 Table Relevance Estimation

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

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

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

Entity Coverage

When selecting a candidate table, the coverage of the tables' entity set is a important factor [Das Sarma et al., 2012, Ahmadov et al., 2015a]. We compute the fraction of the seed table's entities covered by candidate table as:

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

We note that the same concept is used in [Das Sarma et al., 2012], where it is referred to as *entity coverage*.

Caption Likelihood

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

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

Column Labels Likelihood

Finally, we estimate the column labels likelihood similar to Lehmberg et al. [2015], who rank tables according to the number of overlapping labels:

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

5.4 Experimental design

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

5.4.1 Data

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

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

5.4.2 Entity-Focused Tables

Recall that we defined an entity-focused table as one that contains only unique entities in its leftmost column (cf. Sect. 5.1). In addition to being an entity-focused table, we require that the table has at least 6 rows and at least additional 3 columns (excluding the entity column). We introduce

Table 5.2: Statistics of table corpus. Constraints mean having ≥ 6 rows and ≥ 4 columns.

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

these constraints so that we can simulate a real-world scenario with sufficient amount of content.

In Table 5.2, we report statistics based on what percentage of cells in the leftmost column contains entities. Let us note here that only those entities are recognized that have a corresponding Wikipedia article. Thus, the reported numbers should be treated as lower bound estimates. It is clear that many tables have an entity focus.

To be able to perform an automated evaluation without any human intervention, we apply the most strict conditions. Out of the tables that contain 100% unique entities in their leftmost column and have at least 6 rows and at least 4 columns (53 K in total, see Table 5.2), we randomly select 1000 tables as our validation set (used for parameter tuning) and another 1000 tables as our test set. We use a different random selection of validation/test tables for row and column population. The validation and test sets are excluded from the table corpus during training. It is important to note that we use all other tables from the corpus when computing statistics, and not only those that classify as entity-focused.

5.4.3 Simulation Process

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

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

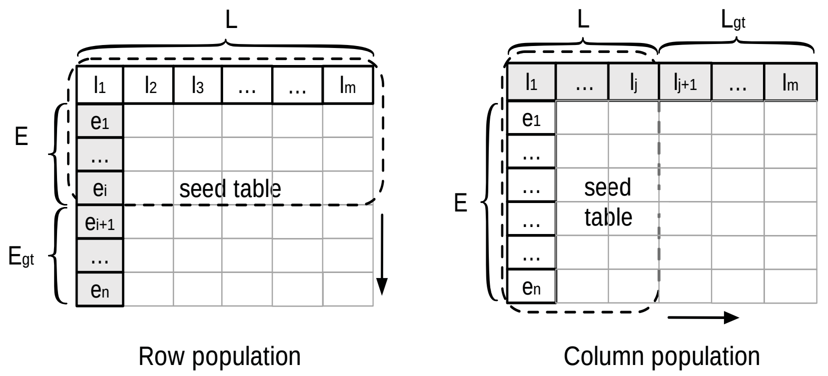


Figure 5.2: Illustration of our evaluation methodology. A part of an existing table is designated as the seed table; the entities/column labels outside the seed table serve as ground truth. The arrows indicate the direction of the population.

remaining columns as ground truth, $\hat{L}$. We use all entities from the table's rows.

See Figure 5.2 for an illustration. Notice that we are expanding in a single dimension at a time; populating both rows and columns at the same time is left for future work.

5.4.4 Matching Column Labels

For the column population task, we are matching string labels (as opposed to unique identifiers). Let us consider `Date` as the ground truth column label. When the suggested labels are compared against this using strict string matching, then `date`, `Dates`, `date:`, etc. would not be accepted as correct, despite being semantically identical. Therefore, we apply some simple normalization steps, on both the ranked and ground truth column labels, before comparing them using strict string matching. When multiple ranked labels are normalized to the same form, only the one with the highest score is retained.

5.4.5 Evaluation Metrics

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

Table 5.3: Candidate selection performance I for the row population task on the validation set. #cand refers to the number of candidate entities. Highest recall values are typeset in boldface.

Method	#Seed entities ($\|E\|$) 1		2		3	
	Recall	#cand	Recall	#cand	Recall	#cand
(A1) Categories (*k*=256)	0.6470	1721	0.6985	2772	0.7282	3678
(A2) Types (*k*=4096)	0.0553	7703	0.0577	8047	0.0585	8225
(B) Table caption (*k*=256)	0.3966	987	0.3961	987	0.3945	987
(C) Table entities (*k*=256)	0.6643	312	0.7212	458	0.7435	589
(B) & (C) (*k*=256)	0.7090	1250	0.7464	1383	0.7626	1505
(A1) & (B) (*k*=256)	0.7642	2671	0.7969	3711	0.8157	4610
(A1) & (C) (*k*=256)	0.8434	1962	0.8885	3118	0.9038	4117
(A1) & (B) & (C) (*k*=256)	0.8662	2880	0.8997	4018	0.9154	5005
(A1) & (B) & (C) (*k*=4096)	**0.9576**	28733	**0.9718**	40171	**0.9787**	49478

Table 5.4: Candidate selection performance II for the row population task on the validation set. #cand refers to the number of candidate entities. Highest recall values are typeset in boldface.

Method	#Seed entities ($\|E\|$) 4		5	
	Recall	#cand	Recall	#cand
(A1) Categories (*k*=256)	0.7476	4507	0.7604	5224
(A2) Types (*k*=4096)	0.0605	8419	0.0600	8551
(B) Table caption (*k*=256)	0.3938	987	0.3929	987
(C) Table entities (*k*=256)	0.7564	689	0.7639	759
(B) & (C) (*k*=256)	0.7732	1599	0.7788	1664
(A1) & (B) (*k*=256)	0.8305	5434	0.8405	6145
(A1) & (C) (*k*=256)	0.9196	5014	0.9285	5773
(A1) & (B) & (C) (*k*=256)	0.9255	5894	0.9329	6645
(A1) & (B) & (C) (*k*=4096)	**0.9811**	58021	**0.9821**	65204

5.5 Evaluation of Row Population

This section presents the evaluation of row population.

5.5.1 Candidate Selection

In Sect. 5.2.1, we have introduced four individual methods to select candidates: entity category (A1) and entity type (A2) from the knowledge base, and table caption (B) and table entities (C) from the table corpus. These methods involve a cut-off threshold parameter *k*; the top-*k* entities are considered as candidates for the subsequent ranking step. A larger k value typically implies higher recall. At the same time, each of the candidate entities will need to be scored, which is a computationally expensive operation. Therefore, we wish to find a setting that ensures high recall, while keeping the number of candidate entities manageably low (to ensure reasonable response time). We use the validation set to explore a range of k values: 2^6, 2^8, 2^{10}, and 2^{12}. For each method, we select the k value that produces the best recall and candidate entity number ratio.

The results are reported in the top block of Table 5.3 and Table 5.4. We observe that more seed entities we have, the better recall gets. This is expected behavior. Out of the two entity properties from the knowledge base, categories and types, categories performs far better. For types, even with $k = 4096$, the recall is still unsatisfactory. This is because many of the DBpedia entities have no ontology type information assigned to them. Moreover, ontology types are more general than categories and result in too many candidates. The best individual method is (C) table entities; it is the most effective (achieves the highest recall) and the most efficient (produces the lowest number of candidates) at the same time.

To further enhance performance, we combine the individual methods. However, we exclude type (A2) from this combination, because of its low performance. We find that all combinations improve over the single methods. This means that they capture complimentary aspects. Combining all three methods (A1+B+C) leads to the best overall performance. The last two lines of Table 5.3 and Table 5.4 show the performance of this combination (A1+B+C) using two different k values. We find that with a high k value (4096), we are able to achieve close to perfect recall. The number of candidates, however, is a magnitude larger than with a low k (256). Motivated by efficiency considerations, we decided not to pay this price and chose to use $k = 256$, which still gives us very high recall.

5.5.2 Entity Ranking

Our entity ranking model is comprised of three components: entity similarity ($P(e|E)$), column labels likelihood ($P(L|e)$), and caption likelihood ($P(c|e)$). Each of these methods involve an interpolation parameter (λ_E, λ_L, and λ_c, respectively). We train these parameters on the validation set, by performing a sweep in 0.1 steps over the [0..1] range. The effect of varying the parameter values is shown in Figure 5.3. It can be seen that the value 0.5 provides the best setting everywhere. We also found that there is very little difference in terms of performance when λ is in the 0.3..0.7 range (hence the choice of showing the 0.1 and 0.9 values on the plots).

We start by discussing the performance of individual components, reported in the top block of Table 5.5 and Table 5.6. The two-component entity simi-

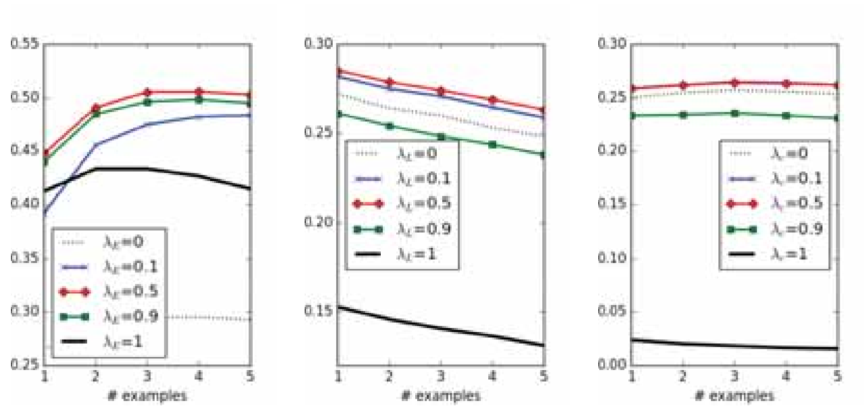


Figure 5.3: Effect of varying the interpolation parameters for $P(e|E)$ (Left), $P(L|e)$ (Middle), and $P(c|e)$ (Right). The plots show MAP scores measured on the validation set.

larity model combines estimated based on the knowledge base and the table corpus (cf. Eq. (5.2)). For the former, we compare three alternatives: using relations of entities, as in [Bron et al., 2013] (A1), and two similarity methods based on outgoing links of entities: WLM (A2), and Jaccard similarity (A3). Out of the three methods, (A1) Relations has the best performance. However, (A3) has only marginally lower retrieval performance, while being computationally much more efficient. Therefore, we choose (A3), when it comes to combining it with the other elements of the entity ranking model. Compared to entity similarity ($P(e|E)$), the other two components (B and C) have much lower performance. The differences (A3) vs. (B) and (A3) vs. (C) are highly significant ($p < 10^{-5}$). This means that the knowledge base contributes more.

Next, we combine the individual components to further enhance performance. The middle block of Table 5.5 and Table 5.6 reports results when two components are used. We find that these combinations improve significantly over the individual methods in all cases ($p < 10^{-5}$). It is interesting to note that while (C) caption likelihood outperforms (B) column labels likelihood in the individual comparison (significantly so for #1..#3 seed entities, $p < 0.001$), the two perform on a par when combined with (A3) entity similarity.

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

Table 5.5: Entity ranking performance I on the test set.

Method	#Seed entities ($\|E\|$)					
	1		2		3	
	MAP	MRR	MAP	MRR	MAP	MRR
Baseline [Bron et al., 2013]	0.3076	0.4967	0.3273	0.5156	0.3404	0.5326
(A1) $P(e\|E)$ Relations ($\lambda = 0.5$)	0.4962	0.6857	0.5469	0.7297	0.5687	0.7415
(A2) $P(e\|E)$ WLM ($\lambda = 0.5$)	0.4674	0.6246	0.5154	0.6901	0.5293	0.6930
(A3) $P(e\|E)$ Jaccard ($\lambda = 0.5$)	0.4905	0.6731	0.5427	0.7086	0.5617	0.7270
(B) $P(L\|e)$	0.2857	0.3558	0.2878	0.3518	0.2717	0.3463
(C) $P(c\|e)$	0.2348	0.2656	0.2366	0.2676	0.2371	0.2656
(A3) & (B)	0.5726	0.7593	0.6108	**0.8055**	0.6189	**0.7879**
(A3) & (C)	0.5743	0.7467	0.6108	0.7749	0.6221	0.7746
(B) & (C)	0.3677	0.4521	0.3715	0.4508	0.3712	0.4455
(A3) & (B) & (C)	**0.5922**	**0.7729**	**0.6260**	0.8000	**0.6339**	0.7849

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

For baseline comparison, we employ the method by Bron et al. [2013], which combines text-based and structure-based similarity. Note that we used only the structure-based part of their method earlier, as (A1); here, we use their approach in its entirety. It requires a keyword query, which we set to be the table caption. We find that our methods substantially and significantly ($p < 10^{-5}$) outperforms this baseline; see the bottom two rows in Table 5.5 and Table 5.6.

One final observation is that performance climbs when moving from a single to two and three seed entities; after that, however, it plateaus. This behavior is consistent across all methods, including the baseline. The phenomena is known from prior work [Bron et al., 2013, Metzger et al., 2014, Pantel et al., 2009].

5.5.3 Analysis

Now that we have presented our overall results, we perform further examination on the level of individual tables. Figure 5.4 shows the average precision (AP) scores for the 1000 test tables, ordered by decreasing score. Statistically, there are 285 tables having $AP = 1$, 193 tables having $0.4 < AP < 0.6$, and 42 tables having $AP = 0$. To understand the reasons behind this, we check the recall of the candidate selection step for these three categories; see Figure 5.5. In this figure, we can observe that higher recall generally leads to better AP. Delving deeper, we compute the average number of tables containing at least one ground truth entity, for each of the three groups. When

Table 5.6: Entity ranking performance II on the test set.

Method	#Seed entities ($\lvert E \rvert$) 4 MAP	4 MRR	5 MAP	5 MRR
Baseline [Bron et al., 2013]	0.3428	0.5290	0.3406	0.5202
(A1) $P(e\|E)$ Relations ($\lambda = 0.5$)	0.5734	0.7294	0.5693	0.7274
(A2) $P(e\|E)$ WLM ($\lambda = 0.5$)	0.5331	0.6861	0.5258	0.6789
(A3) $P(e\|E)$ Jaccard ($\lambda = 0.5$)	0.5662	0.7098	0.5609	0.7058
(B) $P(L\|e)$	0.2651	0.3365	0.2585	0.3336
(C) $P(c\|e)$	0.2350	0.2614	0.2343	0.2602
(A3) & (B)	0.6182	0.7755	0.6108	**0.7689**
(A3) & (C)	0.6211	0.7668	0.6156	0.7447
(B) & (C)	0.3688	0.4408	0.3667	0.4378
(A3) & (B) & (C)	**0.6348**	**0.7800**	**0.6310**	0.7630

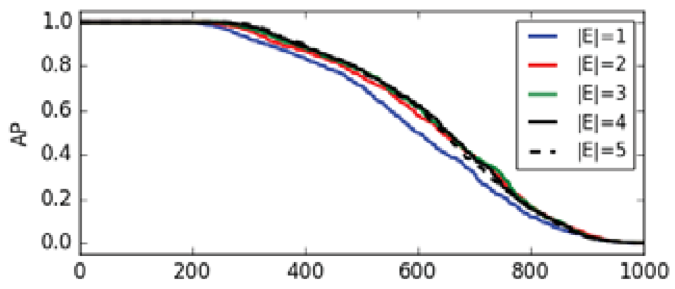


Figure 5.4: Performance of individual tables, ordered by decreasing Average Precision, for row population.

$AP = 0$, the number is 18, for $0.4 < AP < 0.6$ it is 79, and for $AP = 1$ it is 127. It appears that we could provide excellent suggestions, when there were enough similar tables to the seed table in the table corpus. However, for tables that are "too unique," we would need alternative methods for suggestions.

5.6 Evaluation of Column Population

This section presents the evaluation of column population. This task relies only on the table corpus; the data set is exactly the same as for row population, see Sect. 5.4.1.

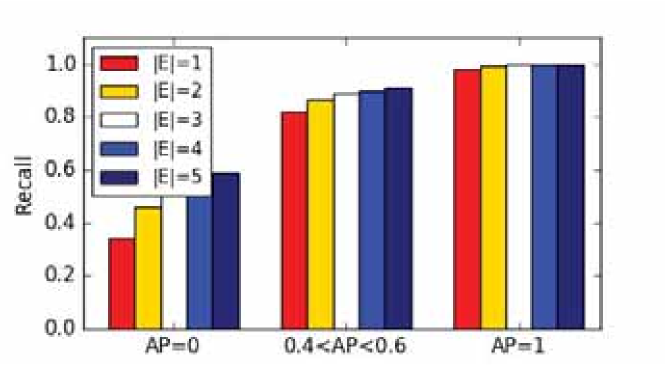


Figure 5.5: Recall of candidate selection against entity ranking performance, for row population.

Table 5.7: Candidate selection performance for the column population task on the validation set.

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

5.6.1 Candidate Selection

In Sect. 5.3.1, we have introduced three individual methods to select candidates: table caption (A), column heading labels (B) and table entities (C). Method (B) actually corresponds to the FastJoin matcher in [Wang et al., 2014]. These methods also involve a cut-off threshold parameter k, for the same reasons we already discussed in Sect. 5.5.1. The results are reported in the top block of Table 5.7. We observe that the more seed labels we have the better recall gets when using labels. We also explore combinations of pairs of methods as well as using all three. We find that all combinations improve over the single methods, and that combining all three methods leads to the best overall performance. Our selected method is the second to last in Table 5.7, motivated by efficiency considerations; for comparison, we also show the performance for $k = 4096$.

5.6.2 Column Label Ranking

Our column label ranking model is comprised of two components: table relevance and label likelihood. For estimating candidate table relevance,

Table 5.8: Column label ranking performance on the test set.

Method	#Seed column labels ($|L|$)					
	1		2		3	
	MAP	MRR	MAP	MRR	MAP	MRR
Baseline [Das Sarma et al., 2012]	0.4413	0.5473	0.4640	0.5535	0.4535	0.5079
(A) Table caption	0.2584	0.3496	0.2404	0.2927	0.2161	0.2356
(B) Column labels	0.2463	0.3676	0.3145	0.4276	0.3528	0.4246
(C) Table entities	0.3878	0.4544	03714	0.4187	0.3475	0.3732
(A) & (B)	0.4824	0.5896	0.4929	0.5837	0.4826	0.5351
(A) & (C)	0.5032	0.5941	0.4909	0.5601	0.4724	0.5132
(B) & (C)	0.5060	0.5954	0.5410	0.6178	0.5323	0.5802
(A) & (B) & (C)	**0.5863**	**0.6854**	**0.5847**	**0.6690**	**0.5696**	**0.6201**

we have three individual methods, using table caption (A), column labels (B), and table entities (C). All methods use the same estimation of label likelihood (cf. Sect. 5.3.3).

We start by discussing the performance of individual methods, which is reported in the top block of Table 5.8. Of the three, method (C) outperforms the other two, and does significantly so ($p < 10^{-5}$). Looking at the tendency of MAP, the increasing number of seed column labels only contributes to method (B). When combining two of the methods, all combinations improve significantly over the individual methods ($p < 10^{-5}$). Out of the three, (B) & (C) performs best in terms of both MAP and MRR. In the end, putting together all three individual methods delivers the best results. Also, this combination (A & B & C) improves significantly over the combination of any two of the methods ($p < 10^{-5}$).

For baseline comparison, we employ the method by Das Sarma et al. [2012]. They consider the "benefits" of adding additional columns, which expressed in Eq. (5.4). We find that our three-component method substantially and significantly ($p < 10^{-5}$) outperforms this baseline. It should be noted that the baseline in [Das Sarma et al., 2012] uses our candidate selection method to make it comparable; this actually performs better than their original approach.

5.6.3 Analysis

Figure 5.6 plots the performance of individual (test) tables, in decreasing order of average precision score. We find that there are 427 tables having $AP = 1$, 122 tables having $0.4 < AP < 0.6$, and 186 tables having $AP = 0$. We examine these three table groups, based on performance, in terms of their corresponding recall values from the candidate selection step. Figure 5.7 shows these values (averaged over all tables that fall in the

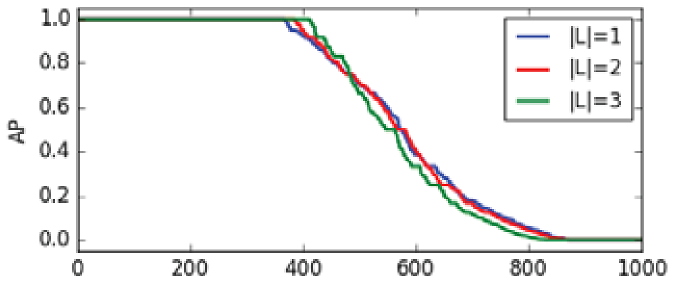


Figure 5.6: Performance of individual tables, ordered by decreasing Average Precision, for column population.

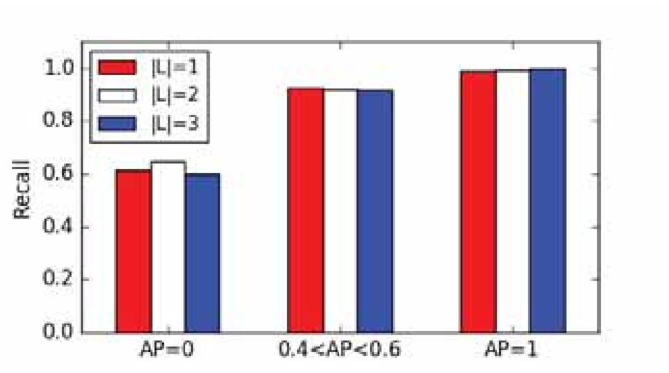


Figure 5.7: Recall of candidate selection against column label ranking performance, for column population.

given performance group). Looking at the number of tables containing at least one ground truth column heading label, it is 204 for $AP = 0$, 403 for $0.4 < AP < 0.6$, and 1114 for $AP = 1$. We can draw similar conclusions here as we did for entity ranking.

5.7 Summary and Conclusions

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

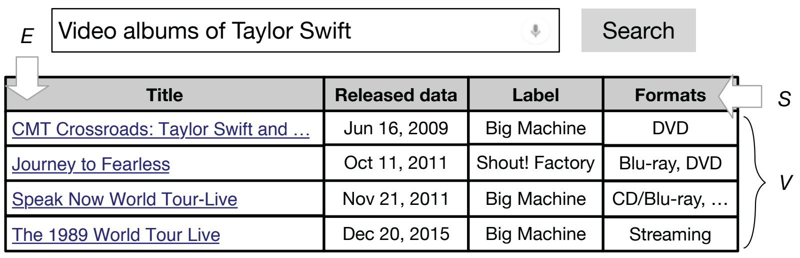
In this study, we have studied one particular type of assistance, for completing row and column headings of a table. In the next chapters, we aim to generating a table responding to natural language queries in Chapter 6 and provide support for filling in value cells in Chapter 7.

Chapter 6

Table Generation

Many information needs revolve around entities, which would be better answered by summarizing results in a tabular format, rather than presenting them as a ranked list. There exist two main families of methods that can return a table as answer to a keyword query. One is to perform table search to find existing tables on the Web. The other is to assemble a table in a row-by-row fashion [Yang et al., 2014] or by joining columns from multiple tables [Pimplikar and Sarawagi, 2012]. However, these methods are limited to returning tables that already exist in their entirety or at least partially (as complete rows or columns). Instead, we aim to answer queries by automatically compiling a table in response to a query. One line of related research is about translating a keyword or natural language query to a structured query language (e.g., SPARQL), which can be executed over a knowledge base [Yahya et al., 2012]. While in principle these techniques could return a list of tuples as the answer, in practice, they are targeted for factoid questions or at most a single attribute per answer entity. More importantly, they require data to be available in a clean, structured form in a consolidated knowledge base.

In this chapter, we introduce and address the task of on-the-fly table generation: given a query, generate a relational table that contains relevant entities (as rows) along with their key properties (as columns). For example, for the query "video albums of Taylor Swift," we can list the albums in a table, as shown in Fig. 6.1. This problem is decomposed into three specific subtasks: (i) core column entity ranking, (ii) schema determination, and (iii) value lookup. We employ a feature-based approach for entity ranking and schema determination, combining deep semantic features with task-specific signals. We further show that these two subtasks are not independent of each other and can assist each other in an iterative manner. For value lookup, we combine information from existing tables and a knowledge base. Using two sets of entity-oriented queries, we evaluate our approach both on the component



Title	Released data	Label	Formats
CMT Crossroads: Taylor Swift and ...	Jun 16, 2009	Big Machine	DVD
Journey to Fearless	Oct 11, 2011	Shout! Factory	Blu-ray, DVD
Speak Now World Tour-Live	Nov 21, 2011	Big Machine	CD/Blu-ray, ...
The 1989 World Tour Live	Dec 20, 2015	Big Machine	Streaming

Figure 6.1: Answering a search query with an on-the-fly generated table, consisting of core column entities E, table schema S, and data cells V.

level and on the end-to-end table generation task.

The chapter is organized as follows. We introduce the task of on-the-fly table generation and propose an iterative table generation algorithm in Sect. 6.1. We develop feature-based approaches for core column entity ranking in Sect. 6.2 and schema determination in Sect. 6.3, and design an entity-oriented fact catalog for fast and effective value lookup in Sect. 6.4. We introduce the queries and experimental setup in Sect. 6.5. We perform extensive evaluation on the component level in Sect. 6.6 and provide further insights and analysis in Sect. 6.7. Section 6.8 concludes this chapter.

6.1 Overview

The objective of on-the-fly table generation is to assemble and return a relational table as the answer in response to a free text query. Formally, given a keyword query q, the task is to return a table $T = (E, S, V)$, where $E = \langle e_1, \ldots e_n \rangle$ is a ranked list of core column entities, $S = \langle s_1, \ldots s_m \rangle$ is a ranked list of heading column labels, and V is an n-by-m matrix, such that v_{ij} refers to the value in row i and column j of the matrix ($i \in [1..n]$, $j \in [1..m]$). Table 6.1 lists the notation used in this chapter. According to the needed table elements, the task boils down to (i) searching core column entities, (ii) determining the table schema, and (iii) looking up values for the data cells. Figure 6.2 shows how these three components are connected to each other in our proposed approach.

6.1.1 Iterative Table Generation Algorithm

There are some clear sequential dependencies between the three main components: *core column entity ranking* and *schema determination* need to be performed before *value lookup*. Other than that, the former two may be conducted independently of and parallel to each other. However, we postulate that better overall performance may be achieved if core column entity rank-

Table 6.1: Notation used in this chapter.

Symbol	Description
q	Query
T	Table
E	Core column entities $E = \langle e_1, \ldots e_n \rangle$
S	Heading column labels $S = \langle s_1, \ldots s_m \rangle$
V	Value matrix

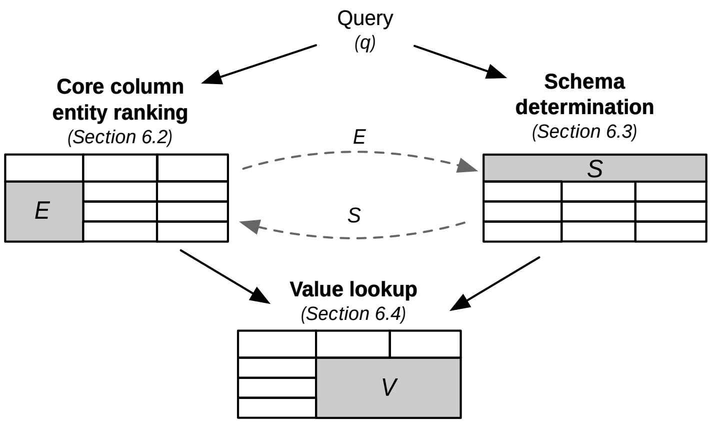


Figure 6.2: Overview of our table generation approach.

ing and schema determination would supplement each other. That is, each would make use of not only the input query, but the other's output as well. To this end, we propose an iterative algorithm that gradually updates core column entity ranking and schema determination results.

The pseudocode of our approach is provided in Algo 1, where *rankEntites*$()$, *rankLabels*$()$, and *lookupValues*$()$ refer to the subtasks of core column entity ranking, schema determination, and value lookup, respectively. Initially, we issue the query q to search entities and schema labels, by *rankEntites*$(q, \{\})$ and *rankLabels*$(q, \{\})$. Then, in a series of successive iterations, indexed by t, core column entity ranking will consider the top-k ranked schema labels from iteration $t-1$ (*rankEntites*(q, S^{t-1})). Analogously, schema determination will take the top-k ranked core column entities from the previous iteration (*rankLabels*(q, E^{t-1})). These steps are repeated until some termination condition is met, e.g., the rankings do not change beyond a certain extent anymore. We leave the determination of a suitable termination condition to future work and will use a fixed number of iterations in our experiments. In the final step of our algorithm, we look up values V using the core col-

Algorithm 1: Iterative Table Generation

```
   Data: $q$, a keyword query
   Result: $T = (E, S, V)$, a result table
1  begin
2      $E^0 \leftarrow rankEntities(q, \{\})$;
3      $S^0 \leftarrow rankLabels(q, \{\})$;
4      $t \leftarrow 0$ ;
5      while $\neg terminate$ do
6          $t \leftarrow t + 1$ ;
7          $E^t \leftarrow rankEntities(q, S^{t-1})$;
8          $S^t \leftarrow rankLabels(q, E^{t-1})$;
9      end
10     $V \leftarrow lookupValues(E^t, S^t)$;
11     return $(E^t, S^t, V)$
12 end
```

umn entities and schema (*lookupValues*(E^t, S^t)). Then, the resulting table (E^t, S^t, V) is returned as output.

6.1.2 Data Sources

Another innovative element of our approach is that we do not rely on a single data source. We combine information both from a collection of existing tables, referred to as the *table corpus*, and from a knowledge base. We shall assume that there is some process in place that can identify relational tables in the table corpus, based on the presence of a core column. We further assume that entities in the core column are linked to the corresponding entries in the knowledge base. The technical details are described in Sect. 6.5. Based on the information stored about each entity in the knowledge base, we consider multiple entity representations: (i) *all* refers to the concatenation of all textual material that is available about the entity (referred to as "catchall" in [Hasibi et al., 2017b]), (ii) *description* is based on the entity's short textual description (i.e., abstract or summary), and (iii) *properties* consists of a restricted set of facts (property-value pairs) about the entity. We will use DBpedia in our experiments, but it can be assumed, without loss of generality, that the above information is available in any general-purpose knowledge base.

6.2 Core column entity ranking

In this section, we address the subtask of *core column entity ranking*: given a query, identify entities that should be placed in the core column of the

Table 6.2: Features used for core column entity retrieval.

Feature	**Iter. (t)**
Term-based matching	
φ_1: $LM(q, e_a)$	≥ 0
Deep semantic matching	
φ_2: $DRRM\_TKS(q, e_d)$	≥ 0
φ_3: $DRRM\_TKS(q, e_p)$	≥ 0
φ_4: $DRRM\_TKS(s, e_d)$	≥ 1
φ_5: $DRRM\_TKS(s, e_p)$	≥ 1
φ_6: $DRRM\_TKS(q \oplus s, e_d \oplus e_p)$	≥ 1
Entity-schema compatibility	
φ_7: $ESC(S, e)$	≥ 1

generated output table. This task is closely related to the problem of ad hoc entity retrieval. Indeed, our initial scoring function is based on existing entity retrieval approaches. However, this scoring can be iteratively improved by leveraging the identified table schema. Our iterative scoring function combines multiple features as ranking signals in a linear fashion:

$$score_t(e, q) = \sum_i w_i \varphi_i(e, q, S^{t-1}) \ , \qquad (6.1)$$

where φ_i is a ranking feature and w_i is the corresponding weight. In the first round of the iteration ($t = 0$), the table schema is not yet available, thus S^{t-1} by definition is an empty list. For later iterations ($t > 0$), S^{t-1} is computed using the methods described in Sect. 6.3. For notational convenience, we shall write S to denote the set of top-k schema labels from S^{t-1}. In the remainder of this section, we present the features we developed for core column entity ranking; see Table 6.2 for a summary.

6.2.1 Query-based Entity Ranking

Initially, we only have the query q as input. We consider term-based and semantic matching as features.

Term-based matching

There is a wide variety of retrieval models for term-based entity ranking [Hasibi et al., 2017b]. We rank document-based entity representations using Language Modeling techniques (cf. Eq. (5.3)). Despite its simplicity, this model has shown to deliver competitive performance [Hasibi et al., 2017b]. Specifically, following Hasibi et al. [2017b], we use the *all* entity representation, concatenating all textual material available about a given entity.

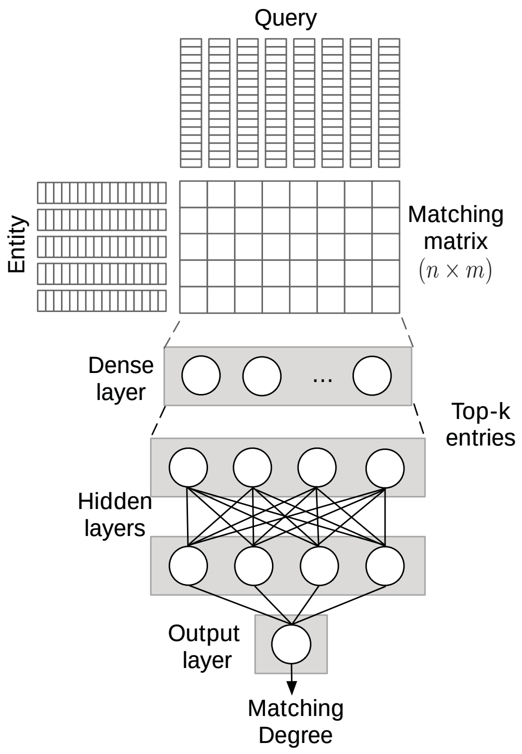


Figure 6.3: Architecture of the DRRM_TKS deep semantic matching method.

Deep semantic matching

We employ a deep semantic matching method, referred to as DRRM_TKS [Fan et al., 2017]. It is an enhancement of DRRM [Guo et al., 2016] for short text, where the matching histograms are replaced with the top-k strongest signals. Specifically, the entity and the query are represented as sequences of embedding vectors, denoted as $e = [w_1^e, w_2^e, ..., w_n^e]$ and $q = [w_1^q, w_2^q, ..., w_m^q]$. An $n \times m$ matching matrix M is computed for their joint representation, by setting $M_{ij} = w_i^e \cdot (w_j^q)^\intercal$. The values of this matrix are used as input to the dense layer of the network. Then, the top-k strongest signals, based on a softmax operation, are selected and fed into the hidden layers. The output layer computes the final matching score between the query and entity. The architecture of DRRM_TKS is shown in Fig. 6.3.

We instantiate this neural network with two different entity representations: (i) using the entity's textual description, e_d, and (ii) using the properties of the entity in the knowledge base, e_p. The matching degree scores computed using these two representations, $DRRM\_TKS(q, e_d)$ and $DRRM\_TKS(q, e_p)$, are used as ranking features φ_2 and φ_3, respectively.

6.2.2 Schema-assisted Entity Ranking

After the first iteration, core column entity ranking can be assisted by utilizing the determined table schema from the previous iteration. We present a number of additional features that incorporate schema information.

Deep semantic matching

We employ the same neural network as before, in Sect. 6.2.1, to compute semantic similarity by considering the table schema. Specifically, all schema labels in S are concatenated into a single string s. For the candidate entities, we keep the same representations as in Sect. 6.2.1. By comparing all schema labels s against the entity, we obtain the schema-assisted deep features $DRRM\_TKS(s, e_d)$ and $DRRM\_TKS(s, e_p)$. Additionally, we combine the input query with the schema labels, $q \oplus s$, and match it against a combined representation of the entity, $e_d \oplus e_p$, where $\oplus$ refers to the string concatenation operation. The resulting matching score is denoted as $DRRM\_TKS$ $(q \oplus s, e_d \oplus e_p)$.

Entity-schema compatibility

Intuitively, core column entities in a given table are from the same semantic class, for example, athletes, digital products, films, etc. We aim to capture their semantic compatibility with the table schema, by introducing a measure called *entity-schema compatibility*.

We compare the property labels of core column entities E against schema S to build the compatibility matrix C. Element C_{ij} of the matrix is a binary indicator between the jth schema label and the ith entity, which equals to 1 if entity e_i has property s_j. To check if an entity has a given property, we look for evidence both in the knowledge base and in the table corpus. Formally:

$$C_{ij} = \begin{cases} 1, & \text{if } match_{KB}(e_i, s_j) \vee match_{TC}(e_i, s_j) \\ 0, & \text{otherwise}, \end{cases}$$

where $match_{KB}(e_i, s_j)$ and $match_{TC}(e_i, s_j)$ are binary indicator functions. The former is true if entity e_i has property s_j in the knowledge base, the latter is true if there exists a table in the table corpus where e_i is a core column entity and s_j is a schema label. Then, the entity-schema compatibility score, which is used as ranking feature φ_7, is computed as follows:

$$ESC(S, e_i) = \frac{1}{|S|} \sum_j C_{ij} .$$

Table 6.3: Features used for schema determination.

Feature			**Iter. (t)**
Column population	φ_1:	$P(s\|q)$	≥ 0
	φ_2:	$P(s\|q, E)$	≥ 1
Deep semantic matching	φ_3:	$DRRM\_TKS(s, q)$	≥ 0
Attribute retrieval	φ_4:	$AR(s, E)$	≥ 1
Entity-schema compatibility	φ_5:	$ESC(s, E)$	≥ 1

For example, for query "Apollo astronauts walked on the Moon" and schema {country, date of birth, time in space, age at first step, ...}, the ESC scores of entities Alan Shepard, Charles Duke, and Bill Kaysing are 1, 0.85, and 0.4, respectively. The former two are Apollo astronauts who walked on the Moon, while the latter is a writer claiming that the six Apollo Moon landings were a hoax.

6.3 Schema determination

In this section, we address the subtask of *schema determination*, which is to return a ranked list of labels to be used as heading column labels (*labels*, for short) of the generated output table. The initial ranking is based on the input query only. Then, this ranking is iteratively improved by also considering the core column entities. Our scoring function is defined as follows:

$$score_t(s, q) = \sum_i w_i \varphi_i(s, q, E^{t-1}), \tag{6.2}$$

where φ_i is a ranking feature with a corresponding weight w_i. For the initial ranking ($t = 0$), core column entities are not yet available, thus E^{t-1} is an empty list. For successive iterations ($t > 0$), E^{t-1} is computed using the methods described in Sect. 6.2. Since we are only interested in the top-k entities, and not their actual retrieval scores, we shall write E to denote the set of top-k entities in E^{t-1}. Below, we discuss various feature functions φ_i for this task, which are also summarized in Table 6.3.

6.3.1 Query-based Schema Determination

At the start, only the input query q is available for ranking labels. To collect candidate labels, we first search for tables in our table corpus that are relevant to the query. We let $\mathcal{T}$ denote the set of top-k ranked tables. Then, the column heading labels are extracted from these tables as candidates: $\mathcal{S} = \{s | s \in T_S, T \in \mathcal{T}\}$. We provide further details below.

Column population

In Chapter 5 we have introduced the task of *column population*: generating a ranked list of column labels to be added to the column headings of a given seed table. We can adapt the method we have developed there by treating the query as if it was the caption of the seed table. Then, the scoring of schema labels is performed according to the following probabilistic formula:

$$P(s|q) = \sum_{T \in \mathcal{T}} P(s|T)P(T|q) \ ,$$

where related tables serve as a bridge to connect the query q and label s. Specifically, $P(s|T)$ is the likelihood of the schema label given table T and is calculated based on the maximum edit distance [Lehmberg et al., 2015], $dist$,[1] between the s and the schema labels of T:

$$P(s|T) = \begin{cases} 1, & \max_{s' \in T_S} dist(s, s') \geq \gamma \\ 0, & \text{otherwise} \ . \end{cases} \tag{6.3}$$

The probability $P(T|q)$ expresses the relevance of T given the query, and is set proportional to the table's retrieval score (here: BM25).

Deep semantic matching

We employ the same neural network architecture as in Sect. 6.2.1 for comparing labels against the query. For training the network, we use our table corpus and treat table captions as queries. All caption-label pairs that co-occur in an existing table are treated as positive training instances. Negative training instances are sampled from the table corpus by selecting candidate labels that do not co-occur with that caption. The resulting matching score, $DRRM\_TKS(s, q)$, is used as feature φ_3.

6.3.2 Entity-assisted Schema Determination

After the initial round, schema determination can be assisted by considering the set of top-k core column entities, E. The set of candidate labels, from before, is expanded with (i) schema labels from tables that contain any of the entities in E in their core column and (ii) the properties of E in the knowledge base.

Entity enhanced column population

We employ a variant of the column population method from Chapter 5 that makes use of core column entities:

$$P(s|q, E) = \sum_{T} P(s|T)P(T|q, E) \ .$$

[1]Note that despite the name used in [Lehmberg et al., 2015], it is in fact a similarity measure.

The schema label likelihood $P(s|T)$ is computed the same as before, cf. Eq. (6.3). The main difference is in the table relevance estimation component, which now also considers the core column entities:

$$P(T|q, E) = \frac{P(T|E)P(T|q)}{P(T)} .$$

Here, $P(T|E)$ is the fraction of the core column entities covered by a related table, i.e., $|T_E \cap E| / |E|$, and $P(T|q)$ is the same as in Sect. 6.3.1.

Attribute retrieval

Attribute retrieval refers to the task of returning a ranked list of attributes that are relevant given a set of entities [Kopliku et al., 2011]. Using the core column entities as input, we employ the method proposed by Kopliku et al. [2011], which is a linear combination of several features:

$$AR(s, E) = \frac{1}{|E|} \sum_{e \in E} \big(match(s, e, T) + drel(d, e) + sh(s, e) + kb(s, e)\big) .$$

The components of this formula are as follows:

- $match(s, e, T)$ compares the similarity between an entity and a schema label with respect to a given table T. We take T to be the table that is the most relevant to the query ($\arg\max_{T \in \mathcal{T}} P(T|q)$). This matching score is the difference between the table match score and shadow match score:

  $$match(s, e, T) = match(e, T) - match(e, shadow(a)) . \qquad (6.4)$$

  The table match score is computed by representing both the entity and table cells T_{xy} as term vectors, then taking the maximum cosine distance between the two:

  $$match(e, T) = max_{T_{xy} \in T} cos(e, T_{xy}) .$$

  For the latter component in Eq. 6.4, the notion of a *shadow area* is introduced: $shadow(a)$ is set of cells in the table that are in the same row with e or are in the same column with the s. Then, the shadow match score is estimated as:

  $$match(e, shadow(a)) = \max_{T_{xy} \in\, shadow(a)} \cos(e, T_{xy}) .$$

- $drel(d, e)$ denotes the relevance of the document d that contains T:

  $$drel(e) = \frac{\#results - rank(d)}{\#results} ,$$

  where $\#results$ is the number of retrieved results for entity e and $rank(d)$ is the rank of document d within this list.

- $sh(s, e)$ corresponds to the number of search results returned by a Web search engine to a query "$\langle s \rangle$ of $\langle e \rangle$," where s and e are substituted with the label and entity, respectively. If the base-10 logarithm of the number of hits exceeds a certain threshold (10^6 in [Kopliku et al., 2011]) then the feature takes a value of 1, otherwise it is 0.
- $kb(s, e)$ is a binary score indicating whether label s is a property of entity e in the knowledge base (i.e., $s \in e_p$).

Entity-schema compatibility

Similar to Sect. 6.2.2, we employ the entity-schema compatibility feature for schema determination as well. As before, C is a compatibility matrix, where C_{ij} denotes whether entity e_i has property s_j. The ESC score is then computed as follows:

$$ESC(s_j, E) = \frac{1}{|E|} \sum_i C_{ij} \ .$$

6.4 Value Lookup

Having the core column entities and the schema determined, the last component in our table generation approach is concerned with the retrieval of the data cells' values. Formally, for each row (entity) $i \in [1..n]$ and column (schema label) $j \in [1..m]$, our task is to find the value V_{ij}. This value may originate from an existing table in our table corpus or from the knowledge base. The challenges here are twofold: (i) how to match the schema label s_j against the labels of existing tables and knowledge base predicates, and (ii) how to deal with the case when multiple, possibly conflicting values may be found for a given cell.

We go about this task by first creating a catalogue $\mathcal{V}$ of all possible cell values. Each possible cell value is represented as a quadruple $\langle e, s, v, p \rangle$, where e is an entity, s is a schema label, v is a value, and p is provenance, indicating the source of the information. The source may be a knowledge base fact or a particular table in the table corpus. An entity-oriented view of this catalog is a filtered set of triples where the given entity stands as the first component of the quadruple: $e_V = \{\langle s, v, p \rangle | \langle e, s, v, p \rangle \in \mathcal{V}\}$. We select a single value for a given entity e and schema label s according to:

$$score(v, e, s, q) = \max_{\substack{\langle s', v, p \rangle \in e_V \\ match(s, s')}} conf(p, q) \ ,$$

where $match(s, s')$ is a soft string matching function (detailed in Sect. 6.5.3) and $conf(p, q)$ is the confidence associated with provenance p. Motivated by the fact that the knowledge base is expected to contain high-quality manually curated data, we set the confidence score such that the knowledge base is always given priority over the table corpus. If the schema label does not match any predicate from the knowledge base, then we chose the value from the table that is the most relevant to the query. That is, $conf(p, q)$ is based on the corresponding table's relevance score; see Sect. 6.6.3 for the details. Notice that we pick a single source for each value rather than aggregating evidence from multiple sources. The reason for that is that on the user interface, we would like to display a single traceable source where the given value originates from.

6.5 Experimental Setup

Queries, dataset, data preprocessing methods and relevance assessments are introduced in this section.

6.5.1 Test Queries

We use two sets of queries in our experiments:

QS-1 We consider list type queries from the DBpedia-Entity v2 test collection [Hasibi et al., 2017b], that is, queries from SemSearch LS, TREC Entity, and QALD2. Out of these, we use the queries that have at least three highly relevant entities in the ground truth. This set contains 119 queries in total.

QS-2 The RELink Query Collection [Saleiro et al., 2017] consists of 600 complex entity-relationship queries that are answered by entity tuples. That is, the answer table has two or three columns (including the core entity column) and all cell values are entities. The queries and corresponding relevance judgments in this collection are obtained from Wikipedia lists that contain relational tables. For each answer table, human annotators were asked to formulate the corresponding information need as a natural language query, e.g., "find peaks above 6000m in the mountains of Peru."

For both sets, we remove stop words and perform spell correction.

6.5.2 Data Sources

We rely on two main data sources simultaneously: the same knowledge base and table corpus as in Chapter 5.

Further, each table in the table corpus is classified as relational or non-relational according to the existence of a core entity column and the size of the table. We set the following conditions for detecting the core column of a table: (i) the core column should contain the most entities compared to other columns; (ii) if there are more than one columns that have the highest number of entities, then the one with lowest index, i.e., the leftmost one, is regarded as the core column; (iii) the core column must contain at least two entities. Tables without a core column or having less than two rows or columns are regarded as non-relational. In the end, we classify the WikiTables corpus into 973,840 relational and 678,931 non-relational tables. Based on a random sample of 100 tables from each category, we find that all the sampled tables are correctly classified.

6.5.3 Schema Normalization

Different schema labels may be used for expressing the same meaning, e.g., "birthday" vs. "day of birth" or "nation" vs. "country." For the former case, where similar terms are used, we employ a FastJoin match [Wang et al., 2014] to normalize the strings (with stopwords removed). Specifically, we take the maximum edit distance as in [Lehmberg et al., 2015] to measure string similarity. When it exceeds a threshold of δ, we regard them as the same label. We set δ as 0.8 which is consistent with [Lehmberg et al., 2015], where headings are matched for table column join. For the latter case, where different terms are used, we consider predicates connecting the same subject and object as synonyms. These pairs are then checked and erroneous ones are eliminated manually. Whenever schema labels are compared in the chapter, we use their normalized versions.

6.5.4 Relevance Assessments

For QS-1, we consider the highly relevant entities as the ground truth for the core column entity ranking task. For the task of schema determination, we annotated all candidate labels using crowdsourcing. Specifically, we used the Figure Eight platform[2] and presented annotators with the query, three example core column entities, and a label, and asked them to judge the relevance of that label on a three point scale: highly relevant, relevant, or non-relevant. Each query-entity-label triple was annotated by at least three and at most five annotators. The labelling instructions were as follows: a label is highly relevant if it corresponds to an essential table column for the given query and core column entities; a label is relevant when it corresponds to a property shared by most core column entities and provides useful information, but it is not essential for the given query; a label is non-relevant otherwise (e.g., hard to understand, not informative, not relevant, etc.). We take the majority vote to decide the relevance of a label. Statistically, we have 7000 triples annotated, and on average, there are 4.2 highly relevant labels, 1.9 relevant labels, and 49.4 non-relevant labels for each query. The Fleiss'

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

Kappa test statistics for inter-annotator agreement is 0.61, which is considered as substantial agreement [Fleiss et al., 1971]. For the value lookup task, we sampled 25 queries and fetched values from the table corpus and the knowledge base. We again set up a crowdsourcing experiment on Figure Eight for annotation. Given a query, an entity, a schema label, a value, and a source (Wikipedia or DBpedia page), three to five annotators were asked to validate if the value can be found and whether it is correct, according to the provided source. Overall, 14,219 table cell values were validated. The total expense of the crowdsourcing experiments was $560.

QS-2: Since for this query set we are given the ground truth in a tabular format, based on existing Wikipedia tables, we do not need to perform additional manual annotation. The main entities are taken as the ground truth for the core column entity ranking task, heading labels are taken as the ground truth for the schema determination task, and the table cells (for a sample of 25 queries) are taken as the ground truth for the value lookup task.

6.5.5 Evaluation Measures

We evaluate core column entity ranking and schema determination in terms of Normalized Discounted Cumulative Gain (NDCG) at cut-off points 5 and 10. The value lookup task is measured by Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR). To test significance, we use a two-tailed paired t-test and write †/‡ to denote significance at the 0.05 and 0.005 levels, respectively.

6.6 Experimental Evaluation

We evaluate the three main components of our approach, core column entity ranking, schema determination, and value lookup, and assess the effectiveness of our iterative table generation algorithm.

6.6.1 Core Column Entity Ranking

We discuss core column entity ranking results in two parts: (i) using only the query as input, and (ii) leveraging the table schema as well.

Query-based Entity Ranking

The results are reported in top block of Table 6.4. The following methods are compared:

LM For term-based matching we use Language Modeling with Dirichlet smoothing, with the smoothing parameter set to 2000, following Hasibi et al. [2017b]. This method is also used for obtaining the candidate set (top 100 entities per query) that are re-ranked by the methods below.

DRRM_TKS We train the deep matching model using 5-fold cross-validation. We use a four-layer architecture, with 50 nodes in the input layer, two hidden layers in the feed forward matching networks, and one output layer. The optimizer is ADAM [Kingma and Ba, 2014], with hinge loss as the loss function. We set the learning rate to 0.0001 and we report the results after 50 iterations.[3] We employ two instantiations of this network, using entity descriptions (e_d) and entity properties (e_p) as input.

Combined We combine the previous three methods, with equal weights, using a linear combination (cf. Eq. (6.1)). Later, in our analysis in Sect. 6.7.2, we will also experiment with learning the weights for the combination.

On the first query set, QS-1, LM performs best of the single rankers. Combining it with deep features results in 16% and 9% relative improvement for NDCG@5 and NDCG@10, respectively. On QS-2, a slightly different picture emerges. The best individual ranker is DRRM_TKS using entity properties. Nevertheless, the Combined method still improves significantly over the LM baseline.

Schema-assisted Entity Ranking

Next, we also consider the table schema for core column entity ranking. The results are presented in the bottom block of Table 6.4. Note that on top of to the three features we have used before, we have four additional features (cf. Table 6.2). As before, we use uniform weight for all features. We report results for three additional iterations, Rounds #1–#3, where the schema is taken from the previous iteration of the schema determination component. Further, we report on an Oracle method, which uses the ground truth schema. In all cases, we take the top 10 schema labels ($k = 10$); we analyze the effect of using different k values in Sect. 6.7.1. These methods are to be compared against the Combined method, which corresponds to Round #0. We find that our iterative algorithm is able to gradually improve results, in each iteration, for both of the query sets and evaluation metrics; with the exception of QS-1 in Round #1, all improvements are highly significant. Notice that the relative improvement made between Round #0 and Round #3 is substantial: 22% and 86% in terms of NDCG@5 for QS-1 and QS-2, respectively.

[3]We also experimented with C-DSSM and DSSM. However, their overall performance was much lower than that of DRRM_TKS for this task.

Table 6.4: Core column entity ranking results. The top block of the table uses only the keyword query as input. The bottom block of the table uses the table schema; Round #1–#3 rely on automatically determined schema, while the Oracle method uses the ground truth schema. Statistical significance for query-based entity ranking is compared against LM, for schema-assisted entity ranking is compared against the Combined method.

	QS-1		**QS-2**	
Method	**NDCG@5**	**NDCG@10**	**NDCG@5**	**NDCG@10**
Query-based Entity Ranking (Round #0)				
LM	0.2419	0.2591	0.0708	0.0823
DRRM_TKS (e_d)	0.2015	0.2028	0.0501	0.0540
DRRM_TKS (e_p)	0.1780	0.1808	**0.1089**‡	**0.1083**‡
Combined	**0.2821**†	**0.2834**	0.0852‡	0.0920†
Schema-assisted Entity Ranking				
Round #1	0.3012	0.2892	0.1232‡	0.1201‡
Round #2	0.3369‡	0.3221‡	0.1307‡	0.1264‡
Round #3	0.3445‡	0.3250‡	0.1345‡	0.1270‡
Oracle	**0.3518**‡	**0.3355**‡	**0.1587**‡	**0.1555**‡

6.6.2 Schema Determination

Schema determination results are presented in two parts: (i) using only the query as input and (ii) also leveraging core column entities.

Query-based Schema Determination

In the top block of Table 6.5 we compare the following three methods:

CP We employ the column population method from Chapter 5 to determine the top 100 labels for each query. Following Lehmberg et al. [2015], the γ parameter for the edit distance threshold is set to 0.8. This method is also used for obtaining the candidate label set (top 100 per query) that is re-ranked by the methods below.

DRRM_TKS We use the same neural network architecture as for core column entity ranking. For training the network, we make use of Wikipedia tables. If an entity and a schema label co-occur in an existing Wikipedia table, then we consider it as a positive pair. Negative training instances are generated by sampling, for each entity, a set of schema labels that do not co-occur with that entity in any existing table. In the end, we generate a total of 10.7M training examples, split evenly between the positive and negative classes.

Table 6.5: Schema determination results. The top block of the table uses only the keyword query as input. The bottom block of the table uses the core column entities as well; Round #1–#3 rely on automatic entity ranking, while the Oracle method uses the ground truth entities. Statistical significance for query-based schema determination is compared against CP, for entity-assisted entity ranking is compared against the Combined method.

	QS-1		QS-2	
Method	**NDCG@5**	**NDCG@10**	**NDCG@5**	**NDCG@10**
Query-based Entity Ranking (Round #0)				
CP	0.0561	0.0675	0.1770	0.2092
DRRM_TKS	0.0380	0.0427	0.0920	0.1415
Combined	**0.0786**†	**0.0878**†	**0.2310**‡	**0.2695**‡
Entity-assisted Schema Determination				
Round #1	0.1676‡	0.1869‡	0.3342‡	0.3845‡
Round #2	0.1775‡	0.2046‡	0.3614‡	0.4143‡
Round #3	0.1910‡	0.2136‡	0.3683‡	0.4350‡
Oracle	**0.2002**‡	**0.2434**‡	**0.4239**‡	**0.4825**‡

Combined We combine the above two methods in a linear fashion, with equal weights (cf. Eq. (6.2)). Later, in our analysis in Sect. 6.7.2, we will also experiment with learning the weights for the combination.

We find that the CP performs better than DRRM_TKS, especially on the QS-2 query set. The Combined method substantially and significantly outperforms both of them, with a relative improvement of 40% and 30% over CP in terms of NDCG@5 on QS-1 and QS-2, respectively.

Entity-assisted Schema Determination

Next, we incorporate three additional features that make use of core column entities (cf. Table 6.3), using uniform feature weights. For the attribute retrieval feature (Sect. 6.3.2), we rely on the Google Custom Search API to get search hits and use the same parameter setting (feature weights) as in Kopliku et al. [2011]. For all features, we use the top 10 ranked entities (and analyze different *k* values later, in Sect. 6.7.1).

The results are shown in the bottom block of Table 6.5. Already Round #1 shows a significant jump in performance compared to the Combined method (corresponding to Round #0). Subsequent iterations results in further improvements, reaching a relative improvement of 243% and 159% for Round #3 in terms of NDCG@5 for QS-1 and QS-2, respectively. Judging

Table 6.6: Value lookup results.

	QS-1		QS-2	
Source	**MAP**	**MRR**	**MAP**	**MRR**
KB	0.7759	0.7990	0.0745	0.0745
TC	0.1614	0.1746	0.9564	0.9564
KB+TC	0.9270	0.9427	0.9564	0.9564

from the performance of the Oracle method, there is further potential for improvement, especially for QS-2.

6.6.3 Value Lookup

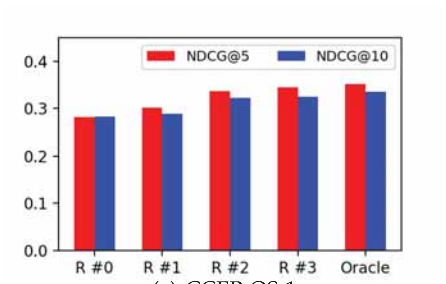
For value lookup evaluation we take the core column entities and schema labels from the ground truth. This is to ensure that this component is evaluated on its own merit, without being negatively influenced by errors that incur earlier in the processing pipeline. In our evaluation, we ignore cells that have empty values according to the ground truth (approximately 12% of the cells have empty values in the Wikitables corpus). The overall evaluation results are reported in Table 6.6. We rely on two sources for value lookup, the knowledge base (KB) and the table corpus (TC). Overall, we reach excellent performance on both query sets. On QS-1, the knowledge base is the primary source, but the table corpus also contributes new values. On QS-2, since all values originate from existing Wikipedia tables, using the knowledge base does not bring additional benefits. This, however, is the peculiarity of that particular dataset. Also, according to the ground truth there is a single correct value for each cell, hence the MAP and MRR scores are the same for QS-2.

6.7 Analysis

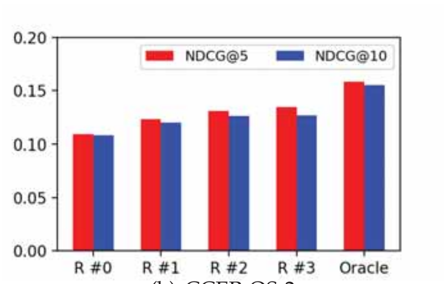
In this section, we conduct further analysis to provide insights on our iterative algorithm and on feature importance.

6.7.1 Iterative Algorithm

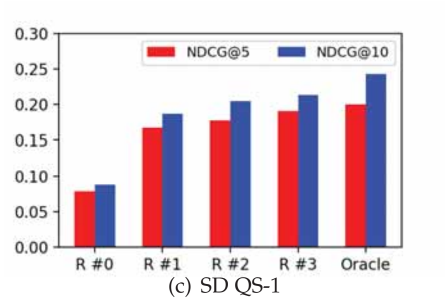
We start our discussion with Fig. 6.4, which displays the overall effectiveness of our iterative algorithm on both tasks. Indeed, as it is clearly shown by these plots, our algorithm performs well. The improvements are the most pronounced when going from Round #0 to Round #1. Performance continues to rise with later iterations, but, as it can be expected, the level of improvement decreases over time. The rightmost bars in the plots correspond to the Oracle method, which represents the upper limit that could



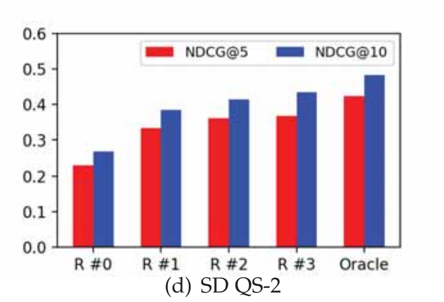
(a) CCER QS-1



(b) CCER QS-2



(c) SD QS-1



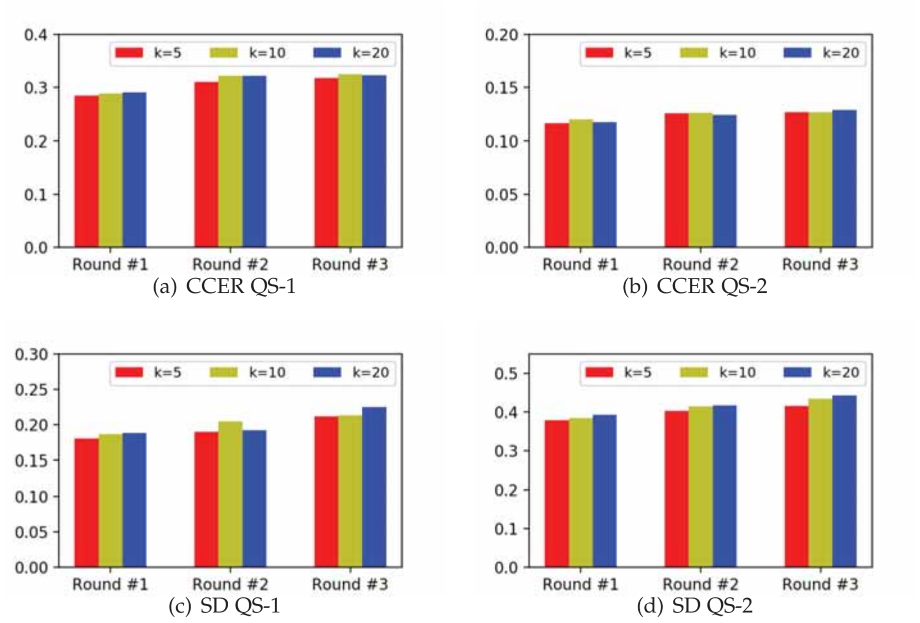
(d) SD QS-2

Figure 6.4: Performance change across iterations for core column entity ranking (CCER) and schema determination (SD).

be achieved, given a perfect schema determination method for core column entity ranking and vice versa. We can observe that for core column entity ranking on QS-1 (Fig. 6.4(a)), has already reached this upper performance limit at iteration #3. For the other task/query set combinations there remains some performance to be gained. It is left for future work to devise a mechanism for determining the number of iterations needed.

Next, we assess the impact of the number of feedback items leveraged, that is, the value of k when using the top-k schema labels in core column entity ranking and top-k entities in schema determination. Figure 6.5 shows how performance changes with different k values. For brevity, we report only on NDCG@10 and note that a similar trend was observed for NDCG@5. We find that the differences between the different k values are generally small, with $k = 10$ being a good overall choice across the board.

To further analyze how individual queries are affected over iterations, Table 6.7 reports the number of queries that are helped (↑), hurt (↓), and remained unchanged (−). We define change as a difference of $\geq$0.05 in terms of NDCG@10. We observe that with the exception of schema determination on QS-2, the number of queries hurt always decreases between successive iterations. Further, the number of queries helped always increases from



(a) CCER QS-1 (b) CCER QS-2 (c) SD QS-1 (d) SD QS-2

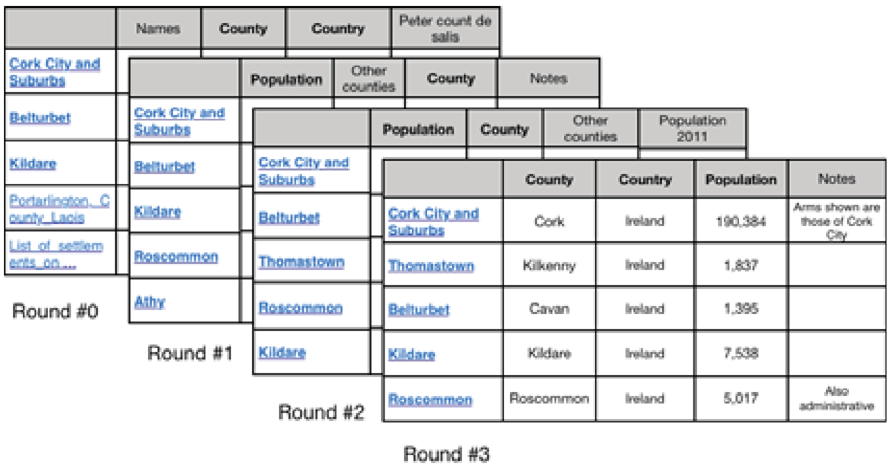
Figure 6.5: Impact of the cutoff parameter k for Core Column Entity Ranking (CCER) and Schema Determination (SD).

Round #1 to #3.

Lastly, we demonstrate how results change over the course of iterations, we show one specific example table in Fig. 6.6 that is generated in response to the query "Towns in the Republic of Ireland in 2006 Census Records."

6.7.2 Parameter Learning

For simplicity, we have so far used all features with equal weights for core column entity ranking (cf. Eq. (6.1)) and schema determination (cf. Eq. (6.2)). Here, we aim to learn the feature weights from training data. In Tables 6.8 and 6.9 we report results with weights learned using five-fold cross-validation. These results are to be compared against the uniform weight settings in Tables 6.4 and 6.5, respectively. We notice that on QS-1, most evaluation scores are lower with learned weights than with uniform weights, for both core column entity ranking and schema determination. This is due to the fact that queries in this set are very heterogeneous [Hasibi et al., 2017b], which makes it difficult to learn weights that perform well across the whole set. On QS-2, according to expectations, learning the weights can yield up to 18% and 21% relative improvement for core column entity ranking and schema determination, respectively.



Round #0

	Names	County	Country	Peter count de salis
Cork City and Suburbs				
Belturbet				
Kildare				
Portarlington, County Laois				
List of settlements on ...				

Round #1

	Population	Other counties	County	Notes
Cork City and Suburbs				
Belturbet				
Kildare				
Roscommon				
Athy				

Round #2

	Population	County	Other counties	Population 2011
Cork City and Suburbs				
Belturbet				
Thomastown				
Roscommon				
Kildare				

Round #3

	County	Country	Population	Notes
Cork City and Suburbs	Cork	Ireland	190,384	Arms shown are those of Cork City
Thomastown	Kilkenny	Ireland	1,837	
Belturbet	Cavan	Ireland	1,395	
Kildare	Kildare	Ireland	7,538	
Roscommon	Roscommon	Ireland	5,017	Also administrative

Figure 6.6: Generated table in response to the query "Towns in the Republic of Ireland in 2006 Census Records." Relevant entities and schema labels are boldfaced.

Table 6.7: The number queries helped (ΔNDCG@10$\geq$0.05), hurt (ΔNDCG@10$\leq$-0.05), and unchanged (remaining) for core column entity ranking (CCER) and schema determination (SD).

	CCER			SD		
QS-1	↑	↓	–	↑	↓	–
Round #0 vs. #1	43	38	38	52	7	60
Round #0 vs. #2	50	30	39	61	5	53
Round #0 vs. #3	49	26	44	59	2	58
QS-2	↑	↓	–	↑	↓	–
Round #0 vs. #1	166	82	346	386	56	158
Round #0 vs. #2	173	74	347	388	86	126
Round #0 vs. #3	173	72	349	403	103	94

6.7.3 Feature Importance

To measure the importance of individual features, we use their average learned weights (linear regression coefficients) across all iterations. The ordering of features for core column entity ranking and QS-1 is: $\varphi_1(0.566) > \varphi_7(0.305) > \varphi_6(0.244) > \varphi_2(0.198) > \varphi_5(0.127) > \varphi_4(0.09) > \varphi_3(0.0066)$. For QS-2 it is: $\varphi_7(0.298) > \varphi_1(0.148) > \varphi_3(0.108) > \varphi_4(0.085) > \varphi_5(0.029) > \varphi_2(-0.118) > \varphi_6(-0.128)$. Overall, we find the term-based matching (Language Modeling) score (φ_1) and our novel entity-schema compatibility

Table 6.8: Core column entity retrieval results with parameters learned using five-fold cross-validation. In parentheses are the relative improvements w.r.t. using uniform weights.

	QS-1		QS-2	
Method	**NDCG@5**	**NDCG@10**	**NDCG@5**	**NDCG@10**
Round #0	0.2523 (-11%)	0.2653 (-6%)	0.1003 (+18%)	0.1048 (+14%)
Round #1	0.2782 (-8%)	0.2772 (-4%)	0.1308 (+6%)	0.1252 (+4%)
Round #2	0.3179 (-6%)	0.3180 (-1%)	0.1367 (+5%)	0.1323 (+5%)
Round #3	0.3192 (-7%)	0.3109 (-4%)	0.1395 (+4%)	0.1339 (+5%)
Oracle	0.3017 (-14%)	0.3042 (-9%)	0.1728 (+9%)	0.1630 (+5%)

Table 6.9: Schema determination results with parameters learned using five-fold cross-validation. In parentheses are the relative improvements w.r.t. using uniform weights.

	QS-1		QS-2	
Method	**NDCG@5**	**NDCG@10**	**NDCG@5**	**NDCG@10**
Round #0	0.0928 (+18%)	0.1064 (+21%)	0.2326 (+1%)	0.2710 (+1%)
Round #1	0.1663 (-1%)	0.2066 (+11%)	0.3865 (+16%)	0.4638 (+12%)
Round #2	0.1693 (-5%)	0.2212 (+8%)	0.3889 (+8%)	0.4599 (+11%)
Round #3	0.1713 (-10%)	0.2321 (+9%)	0.3915 (+6%)	0.4620 (+6%)
Oracle	0.1719 (-14%)	0.2324 (-5%)	0.4678 (+10%)	0.5307 (+10%)

score (φ_7) to be the most important features for core column entity ranking. Turning to schema determination, on QS-1 the ordering is: $\varphi_5(0.23) > \varphi_3(0.076) > \varphi_1(-0.035) > \varphi_2(-0.072) > \varphi_4(-0.129)$. For QS-2 it is: $\varphi_5(0.27) > \varphi_4(0.181) > \varphi_1(0.113) > \varphi_3(0.018) > \varphi_2(-0.083)$. Here, entity-schema compatibility (φ_5) is the single most important feature on both query sets.

6.8 Summary and Conclusions

We have introduced the task of on-the-fly table generation, which aims to answer queries by automatically compiling a relational table in response to a query. This problem is decomposed into three specific subtasks: (i) core column entity ranking, (ii) schema determination, and (iii) value lookup. We have employed a feature-based approach for core column entity ranking and schema determination, combining deep semantic features with task-specific signals. We have further shown that these two subtasks are not independent of each other and have developed an iterative algorithm, in which the two reinforce each other. For value lookup, we have entity-oriented fact catalog, which allows for fast and effective lookup from multiple sources. Using two sets of entity-oriented queries, we have demonstrated the effectiveness of our method.

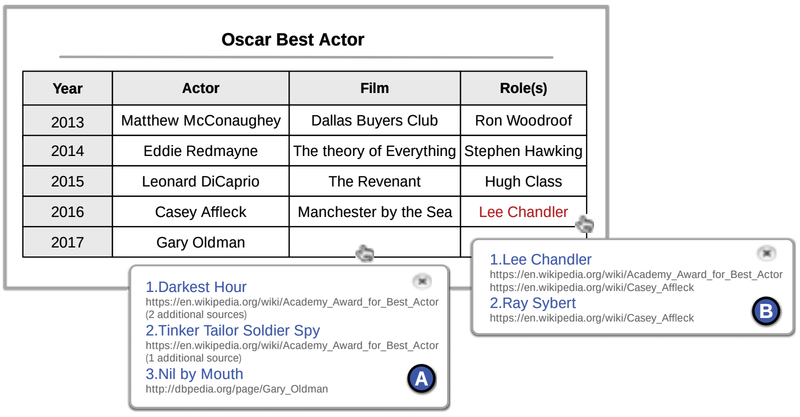
In this chapter, the main focus of our attention was on the core column entity ranking and schema determination subtasks, while for the value finding we presented a simple but effective solution. It should, however, be noted that we considered a somewhat simplified version of the value finding task. For example, we do not deal with the cases where a cell has multiple valid values or when a cell does not have a value and should be left empty. Therefore, we take a closer look at the value finding problem in the next chapter.

Chapter 7

Auto-completion for Data Cells in Relational Tables

Previsouly, we have developed methods for assisting users in the labor-intensive process of table creation by helping them to retrieve existing tables (Chapter 3 and 4), augment tables with data (Chapter 5), and even automatically generate entire tables (Chapter 6). In this chapter, we address the task of finding table cell values with supporting evidence, to support users in the labor-intensive process of creating relational tables. This problem falls in the category of *data augmentation* (also referred to as *data imputation* [Ahmadov et al., 2015a]), which is concerned with extending a given seed table with more data. Examples of data augmentation include populating relational tables with additional rows (entities) and column headings (attributes) (Chapter 5), and automatically finding missing values for data cells [Ahmadov et al., 2015a, Yakout et al., 2012]. This latter task is exactly what we undertake in the current chapter. Figure 7.1 provides an illustration of two scenarios: (A) when finding missing cell values and (B) when validating existing values. We present the CellAutoComplete framework to tackle several novel aspects of this problem, including: (i) enabling a cell to have multiple, possibly conflicting values, (ii) supplementing the predicted values with supporting evidence, (iii) combining evidence from multiple sources, and (iv) handling the case where a cell should be left empty. Our framework makes use of a large table corpus and a knowledge base as data sources, and consists of preprocessing, candidate value finding, and value ranking components. Using a purpose-built test collection, we show that our approach is 40% more effective than the best baseline.

The chapter is organized as follows. We present the CellAutoComplete framework for finding cell values in relational tables in Sect. 7.1. We detail the candidate value finding in Sect. 7.2, and value ranking in Sect. 7.3. Specific novel technical contributions include the heading-to-heading and heading-to-predicate matching components in Sect. 7.2.1 and Sect. 7.2.2) as



Oscar Best Actor

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

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

A

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

B

Figure 7.1: Envisioned user interface. By clicking on a table cell, the user receives a ranked list of suggested values along with supporting sources. Case (A) is to find the value for an empty cell. Case (B) is to check/verify an existing value.

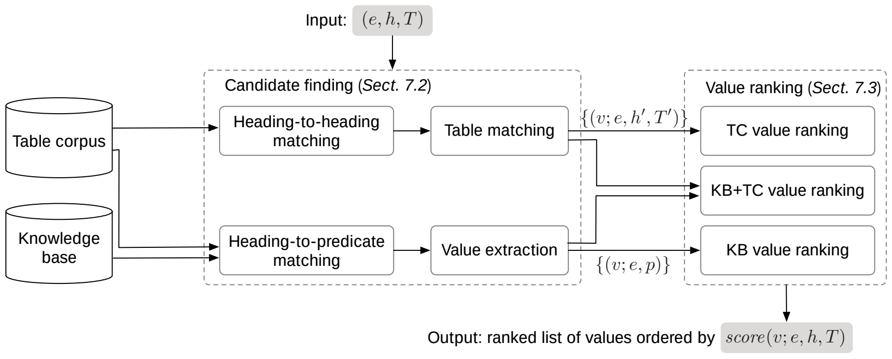

Input: (e, h, T)
Candidate finding (Sect. 7.2)
Value ranking (Sect. 7.3)
Table corpus
Knowledge base
Heading-to-heading matching
Table matching
Heading-to-predicate matching
Value extraction
$\{(v; e, h', T')\}$
$\{(v; e, p)\}$
TC value ranking
KB+TC value ranking
KB value ranking
Output: ranked list of values ordered by $score(v; e, h, T)$

Figure 7.2: Overview of the CellAutoComplete framework.

well as the features designed for combining evidence from multiple sources and for predicting empty values in Sect. 7.3.3. We develop a purpose-built test collection based on Wikipedia tables in Sect. 7.4 and perform an extensive experimental evaluation in Sect. 7.5. Our experiments show that our LTR approach substantially outperforms existing data augmentation techniques. Section. 7.6 concludes this chapter.

7.1 Problem Statement and Overview

We address the task of automatically finding the values of cells in relational tables. A table is said to be relational if it describes a set of entities in its core

Table 7.1: Notation used in this chapter.

Symbol	Description
T	Input table
e	Entity
h	Heading label
v	Value

column (typically, the leftmost column) and the attributes of those entities in additional columns. We shall assume that entities in the core column of each table have been identified and linked to a knowledge base. These annotations may be supplied manually (e.g., tables in Wikipedia) or can be obtained automatically [Efthymiou et al., 2017, Ritze et al., 2015]. The additional (attribute) columns are identified by their heading labels.

Formally, given an input table T, we seek to find the value of the cell that is identified by the row with entity e (in the core column) and the column with heading label h. The output is a ranked list of values v, where the ranking of values is defined by a scoring function $score(v; e, h, T)$. Table 7.1 lists the notation used in this paper.

Our approach, shown in Fig. 7.2, has two main components. First, we identify candidate values from two sources, a table corpus and a knowledge base. Second, these values are ranked based on their likelihood of being correct, and the top-K ranked values are presented to the user as auto-complete suggestions. Note that it is a design decision for us to keep the user in the loop and let her make a judgment call on the appropriateness of a suggestion by considering the supporting evidence.

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

7.2 Candidate Value Finding

In this section, we address the problem of identifying candidate values for a given target cell in table T, identified by the target entity e and target heading label h. Candidate value finding is a crucial step as the recall of the end-to-end task critically depends on it. We gather candidate values from two sources: a table corpus (Sect. 7.2.1) and a knowledge base (Sect. 7.2.2). Novel contributions in this part include the heading-to-heading and heading-to-predicate matchings, and the TMatch table matching approach.

7.2.1 Table Corpus

Our goal is to locate tables from the corpus that contain the target entity and attribute (heading label) pair. We assume a setting where entities in the core table columns have been linked to knowledge base (cf. Sect. 7.4.1). With that, it is easy to find the tables that contain the target entity (with high confidence). The matching of heading labels, however, is not that straightforward: the same meaning may be expressed using different labels (e.g., "established" vs. "founded"), while the same label can mean different things depending on the table's context (e.g., the column label "played" may refer, among others, to the number of games played, to the date of a game, or the name of the opponent). Therefore, we need to perform a matching between heading labels. Naively considering all candidate tables that mention the entity and heading is not sufficient; additionally, we should also take into account their semantic similarity to the input table, referred to as the problem of *table matching* (cf. Chapter 4).

Heading-to-Heading Matching

For a given heading label h, we wish to identify additional heading labels that have the same meaning (i.e., refer to the same entity attribute). This is closely related to the problem of schema matching [Dong and Halevy, 2005]. The main idea is that if two tables T_a and T_b contain the same value v for a given entity e in columns with headings h_a and h_b, respectively, then h_a and h_b *might* mean the same. Sharing a value does not always mean the equivalence of heading labels. Nevertheless, the intuition is that the more often it happens, the more likely it is that h_a and h_b refer to the same entity attribute. We capture this intuition in the following formula:

$$P(h'|h) = \frac{n(h',h)}{\sum_{h''} n(h'',h)} , \tag{7.1}$$

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

Table Matching

All tables in the table corpus that contain (i) the target entity e and (ii) the target heading h or any related heading label h' ($n(h',h) > 0$), are considered as candidates. As we explained above, not all these tables are actually good candidates. Therefore, we estimate the semantic similarity between the input table T and a candidate table T', $score(T,T')$. This table matching score later will be utilized as a confidence estimate in a subsequent value ranking step (in Sect. 7.3). We present two feature-based learning methods for table matching. We start with a state-of-the-art approach, InfoGather. Then, we

Table 7.2: Overview of table ranking features used in TMatch.

Group / Feature	**Source**
Table features	
Number of rows in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
Number of columns in the table	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
Number of empty table cells	[Cafarella et al., 2008a, Bhagavatula et al., 2013]
Table caption IDF	[Qin et al., 2010]
Table page title IDF	[Qin et al., 2010]
Number of in-links to the page embedding the table	[Bhagavatula et al., 2013]
Number of out-links from the page embedding the table	[Bhagavatula et al., 2013]
Number of page views	[Bhagavatula et al., 2013]
Inverse of number of tables on the page	[Bhagavatula et al., 2013]
Ratio of table size to page size	[Bhagavatula et al., 2013]
Matching features	
InfoGather page title IDF similarity score	[Yakout et al., 2012]
InfoGather heading-to-heading similarity	[Yakout et al., 2012]
InfoGather column-to-column similarity	[Yakout et al., 2012]
InfoGather table-to-table similarity	[Yakout et al., 2012]
MSJE heading matching score	[Lehmberg et al., 2015]
Nguyen et al. [2015] heading similarity	[Nguyen et al., 2015]
Nguyen et al. [2015] table data similarity	[Nguyen et al., 2015]
Schema complement schema benefit score	[Das Sarma et al., 2012]
Schema complement entity overlap score	[Das Sarma et al., 2012]
Entity complement entity relatedness score	[Das Sarma et al., 2012]

introduce TMatch, which extends InfoGather with a rich set of features from the literature.

InfoGather InfoGather [Yakout et al., 2012] measures element-wise similarities across four table elements (table data, column values, page title, and heading labels), and combines them in a linear fashion:

$$score(T, T') = \sum_{x} w_x \times sim(T_x, T'_x) ,$$

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

TMatch We extend the four element-wise matching scores of InfoGather with a number of additional matching, which are summarized in Table 7.2.

We use Random Forests regressor as our machine-learned model. We distinguish between two main groups of features. The first group of features (top block in Table 7.2) aim to characterize an individual table and are associated with its quality and importance. These features are computed for both the input and candidate tables. The second group of features (bottom block in Table 7.2) measures the degree of matching between the input and candidate tables. In the interest of space, we present a high-level description of these features and refer to the original publications for details.

- The Mannheim Search Join Engine (MSJE) [Lehmberg et al., 2015] measures the similarity between the headings of two tables by creating an edit distance graph between the input and candidate tables' heading terms. Then, the *maximum weighted bipartite matching score* is computed on this graph's adjacency matrix.
- Nguyen et al. [2015] consider the table headings and table data for matching. Specifically, heading similarity is computed by solving the *maximum weighted bipartite sub-graph problem* [Anan and Avigdor, 2007]. Data similarity is measured by representing each table column as a binary term vector, and then taking the cosine similarity between the most similar column pairs.
- Das Sarma et al. [2012] compute the matching score by aggregating the benefits of adding additional headings columns and entities from the candidate table to the input table.

7.2.2 Knowledge Base

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

Heading-to-Predicate Matching

Both table heading labels and knowledge base predicates represent entity attributes, but these are often expressed differently, making string matching insufficient. Similarly to how it was done in heading-to-heading matching, we capitalize on the observation that if entity e has value v for predicate p in the KB, and the same entity has value v in the heading column h of many tables, then p and h are likely to mean the same (more precisely, h is a string label that corresponds to the semantic relation p). This idea is illustrated in Fig. 7.3. The similarity between heading label (h) and predicate (p) is

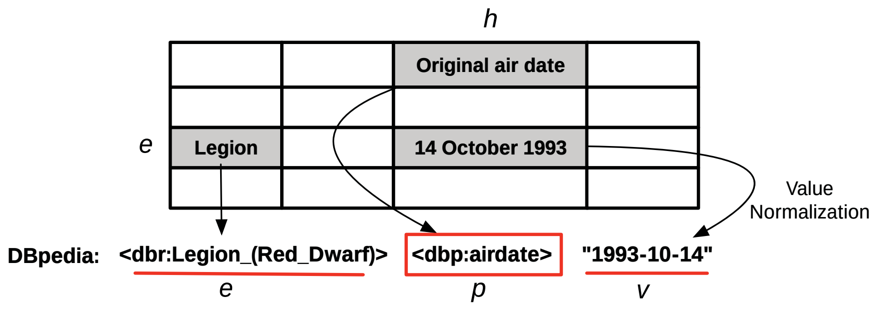


Figure 7.3: Illustration of table heading label to KB predicate matching. Notice that value normalization is also involved.

Table 7.3: Examples of heading label and predicate matches.

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

computed according to the conditional probability $P(p|h)$:

$$P(p|h) = \frac{n(h,p)}{\sum_{p'} n(h,p')} , \tag{7.2}$$

where $n(h,p)$ denotes the times of h and p indicate the same value in the corpus. Table 7.3 lists some examples.

Value Extraction

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

7.3 Value Ranking

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

There are two main challenges we need to deal with. One is how to handle `Empty` values, i.e., quantify our confidence in that the given cell should be left empty. Another is how to combine evidence across multiple sources, specifically, a knowledge base and a table corpus. We start by considering each source individually in Sects. 7.3.1 and 7.3.2, and then combine the two in a feature-based learning approach in Sect. 7.3.3. Novel contributions in this part include the source-specific value finding methods as well as the three groups of features, each designed with a specific intuition in mind: (i) quantifying the support each source has for a given value, (ii) dealing with `Empty` values, and (iii) effectively prioritizing values from semantically more related tables in a table corpus.

7.3.1 Knowledge Base

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

$$score(v; e, h, T) = \begin{cases} \arg\max_p score(p, h), & v \neq \texttt{Empty} \\ \gamma, & v = \texttt{Empty}\,, \end{cases} \tag{7.3}$$

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

- We take the edit distance between the column heading and the (label of the) predicate[1] (which we referred to as soft matching in Sect. 6.3).

  $$score_{ED}(p, h) = 1 - \frac{dist(p, h)}{\max(|p|, |h|)}\,, \tag{7.4}$$

  where $dist()$ represents the minimum number of single-character edit operations (i.e., insertion, deletion, or substitution) needed to transform one string into another.

- We use the conditional probability $P(p|h)$ (cf. Eq. (7.2)).

7.3.2 Table Corpus

A given candidate value may have multiple supporting tables in the corpus. We formulate two evidence combination strategies. One method is to consider a single table that best matches semantically the input table, similarly

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

to [Ahmadov et al., 2015a]. Another method is to consider multiple tables but weigh them according on their semantic similarity to the input table, an idea in line with [Yakout et al., 2012, Zhang and Chakrabarti, 2013]. Both methods are based on the notion of *table matching*, which we described in Sect. 7.2.1. One important addition, compared to prior approaches, is that we also consider which heading h' of the candidate table T' matches best the target heading h (Sect. 7.3.2). As we will show in our experiments, this has a positive effect.

Top-ranked Table

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

$$score(v,e,h,T) = \arg\max_{T'} score(T',T) \times \big(\max_{h' \in T'} sim(h',h)\big) \; . \qquad (7.5)$$

All Tables

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

$$score(v,e,h,T) = \sum_{T'} \Big(score(T',T) \times \max_{h' \in T'} sim(h',h)\Big) \; , \qquad (7.6)$$

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

Heading Label Similarity

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

- *Uniform*: we set similarity to a fixed value (e.g., 1). This way the similarity of headings is not considered at all in Eqs. (7.5) and (7.6). (The uniform estimator will merely serve as a baseline, to evaluate the benefits of incorporating heading similarity.)
- *Edit distance*: We use the edit distance between h and h' (as in Eq. (7.4), but replacing p with h').
- *Mapping probability*: we use the conditional probability $P(h'|h)$ as defined in Eq. (7.1).
- *Label2Vec*: We employ the skip-gram model of Word2vec [Mikolov et al., 2013] to train heading label embeddings on the table corpus. Then, $sim(h',h)$ is taken to be the cosine similarity between the embedding vectors of h' and h, respectively.

Table 7.4: Features for value ranking I. e and h denote the entity and heading column, respectively in the input table T, while h' is the best matching column (based on mapping probability) in the candidate table T'.

Group / Feature	Description	Source	Value
Feature group I			
IS_TC	Whether the value comes from the table corpus ($v \in V_{TC}$)	TC	$\{0,1\}$
IS_KB	Whether the value comes from the knowledge base ($v \in V_{KB}$)	KB	$\{0,1\}$
EDITDIST_PH	Predicate-to-heading edit distance ($score_{ED}(p,h)$)	KB, TC	$[0,1]$
MAPPINGPROB_PH	Predicate-to-heading mapping probability ($P(p\|h)$)	KB, TC	$[0,1]$
EDITDIST_HH	Heading-to-heading edit distance ($score_{ED}(h',h)$)	TC	$[0,1]$
MAPPINGPROB_HH	Heading-to-heading mapping probability ($P(h'\|h)$)	TC	$[0,1]$
TMATCH_NUM	Number of matching tables, using TMatch scorer ($\|\{T' : score_{HCF}(T',T) > 0\}\|$)	TC	$[0,\infty)$
TMATCH_{MAX,AVG,SUM}	Aggregated table matching scores, using TMatch scorer ($aggr_{T'}[score_{HCF}(T',T)]$)	TC	$[0,\infty)$

7.3.3 Combination of Evidence

We combine evidence from multiple sources using a feature-based approach. Table 7.6 summarizes our features. Additionally, we use the same table quality/importance features as for table matching (cf. top block in Table 7.2).

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

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

Table 7.5: Features for value ranking II. e and h denote the entity and heading column, respectively in the input table T, while h' is the best matching column (based on mapping probability) in the candidate table T'.

Group / Feature	**Description**	**Source**	**Value**
Feature group II			
NUM_E	Number of times entity e appears in the table corpus	TC	$[0, \infty)$
NUM_H	Number of times heading h appears in the table corpus	TC	$[0, \infty)$
NUM_EH	Number of times entity e and heading h co-occur in the table corpus	TC	$[0, \infty)$
EMPTY_RATE	Fraction of empty cells in column h in the table corpus	TC	$[0, 1]$
PH_NUM	Number of predicate-to-heading matches ($\|\{p : P(p\|h) > 0\}\|$)	KB, TC	$[0, \infty)$
HH_NUM	Number of heading-to-heading matches ($\|\{h' : P(h'\|h) > 0\}\|$)	TC	$[0, \infty)$
PH_{MAX,AVG,SUM}	Aggregated number of predicate-to-heading matches ($aggr_p[count(p, h)]$)	KB, TC	$[0, \infty)$
HH_{MAX,AVG,SUM}	Aggregated number of heading-to-heading matches ($aggr_{h'}[count(h', h)]$)	TC	$[0, \infty)$

edge base and in the table corpus.[2]

Feature group III aims for capturing the semantic relatedness between the input table and candidate table where the value is taken from. The matches between the input table and candidate tables are captured in the number of matching tables (TMATCH_NUM) as well as aggregates over the table matching scores (TMATCH_*). Additionally, we consider the value scoring mechanism devised specifically for TC (cf. Sect. 7.3.2), which involves a table matching method (InfoGather (IG) or TMatch), heading similarity (edit distance (ED), mapping probability (MP), or Label2vec (L2V)), and an aggregator (MAX, AVG, or SUM). All possible combinations yield a total of 18 features. For example, SCORE_IG_ED_SUM corresponds to Eq. (7.6) using InfoGather for table matching and edit distance heading similarity, and SCORE_TMATCH_L2V_MAX corresponds to Eq. (7.5) using TMatch table matching, and Label2Vec heading similarity.

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

Table 7.6: Features for value ranking III. e and h denote the entity and heading column, respectively in the input table T, while h' is the best matching column (based on mapping probability) in the candidate table T'.

Group / Feature	Description	Source	Value
Feature group III			
SCORE_IG_ED_{MAX,AVG,SUM}	Aggregated value score using InfoGather table matching with edit distance	TC	$[0, \infty)$
SCORE_IG_MP_{MAX,AVG,SUM}	Aggregated value score using InfoGather table matching with mapping probability	TC	$[0, \infty)$
SCORE_IG_L2V_{MAX,AVG,SUM}	Aggregated value score using InfoGather table matching with Label2vec	TC	$[0, \infty)$
TMATCH_ED_{MAX,AVG,SUM}	Aggregated value score using TMatch table matching with edit distance	TC	$[0, \infty)$
TMATCH_MP_{MAX,AVG,SUM}	Aggregated value score using TMatch table matching with mapping probability	TC	$[0, \infty)$
TMATCH_L2V_{MAX,AVG,SUM}	Aggregated value score using TMatch table matching with Label2vec	TC	$[0, \infty)$

7.4 Experimental Setup

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

7.4.1 Data Sources

We use two main data sources:

Table Corpus (TC) The WikiTables corpus [Bhagavatula et al., 2015] is extracted from Wikipedia and contains 1.6M high-quality tables. Following Chapter 3, we select the core column by taking the one among the left-most 2 columns with the highest entity rate. Based on a sample of 100 tables, this method has an accuracy of over 98%. There are 755k relational tables in the corpus that have a core column where 80% of the cell values are entities. Since tables are from Wikipedia, the mentioned entities have been explicitly marked up.

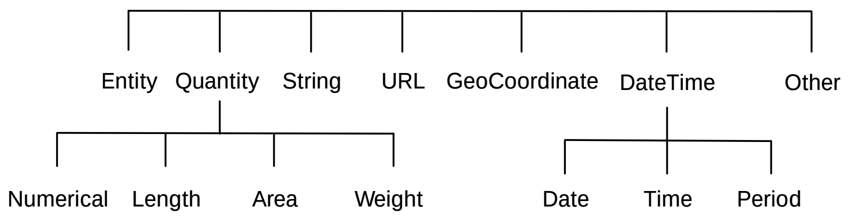


Figure 7.4: Value data type taxonomy used in this paper.

Knowledge Base (KB) DBpedia is a general-purpose knowledge base that is a central hub in the Linked Open Data cloud. Specifically, we use the 2015-10 version and restrict ourselves to entities that have at least a short description (abstract), amounting to a total of 4.6M entities.

7.4.2 Column Data Type Detection

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

Value Data Type Taxonomy

Different value data type taxonomies have been proposed in the literature, see, e.g., [Ritze et al., 2015, Yin et al., 2011]. We build on and extend the taxonomy by Yin et al. [2011], who consider seven value data types in the context of fact finding from web tables: string, date/time, numerical, duration, length, area, and weight. Knowledge bases also have their own data type taxonomies, e.g., DBpedia has 25 data types.[3] Informed by these, we introduce a two-layer value data type taxonomy, which is shown in Figure 7.4. We manually map the data types of the knowledge base (here: DBpedia) to our value data type taxonomy.[4]

Cell Value Type Classification

Following standard practice [Ritze et al., 2015], we design a rule-based method for classifying cell values into our value data type taxonomy. For example, DateTime values are identified based on the cell values matching

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

[4]The mappings will be made available in an online appendix.

given patterns and on certain terms appearing in the column heading label (such as "year," "birth," "date," "founded," "created," or "built"). The complete set of rules will be released in an online appendix upon acceptance.

Evaluation

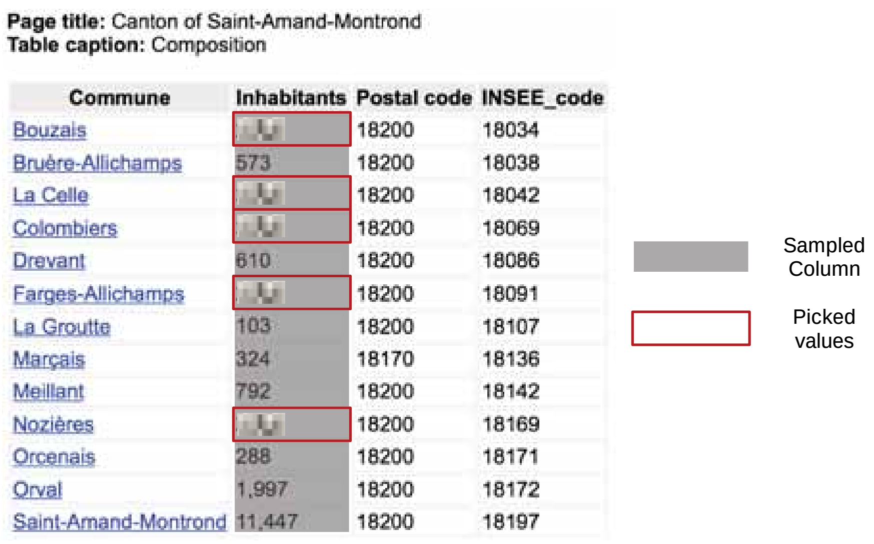
We obtain the distribution of column data types based on a sample of 100K relational tables from the table corpus. The results, ordered by frequency, are as follows: (1) Quantity: 269,260 cells (43.2%), (2) Entity: 200,637 cells (32.2%), (3) String: 76,259 cells (12.2%), (4) Other: 51,999 cells (8.3%),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

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

7.4.3 Value Normalization

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

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


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

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

Figure 7.5: Illustration of sampling values. Firstly, the column of "Inhabitants" is the sampled as the numerical column. Then, five values in this column are picked as the test collection values.

Table 7.7: Summary of our test collection based on 1000 cells.

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

7.4.4 Test Collection

We create a test collection for value finding based on a sample of existing tables from the table corpus. (These test tables are excluded from our index and when computing statistics.) Specifically, we perform stratified sampling according to the four main column data types: Entity, Quantity, String, and DateTime. For each data type, we first randomly select 50 columns, each from a different table, where there is at least 80% agreement on the column data type according to the majority vote method (cf. Sect. 7.4.2). We further require that the table has at least 5 rows and 3 columns, and the respective heading label is at least 4 characters long. From each sampled table column, 5 specific cells are picked randomly. This way, our test collection consists of $4 \times 50 \times 5 = 1000$ cells for which we are trying to find values. These input tables are then excluded from the collection. See Fig. 7.5 for an illustration.

Relevance assessments were collected via crowdsourcing using the Figure

Table 7.8: Value finding performance with empty cells excluded (Top) and included (Bottom). Significance for line i ($i > 1$) is tested against the best method in lines $1..i-1$.

Source	Method	Sources used KB	Sources used TC	Empty excluded NDCG@5	Empty excluded NDCG@10
Single-source	KBLookupED	✓		0.2635	0.2652
	InfoGather, top, UNI		✓	0.4563‡	0.4710‡
	InfoGather, top, L2V		✓	0.4868	0.4978
	TMatch, top, UNI		✓	0.4744‡	0.4873‡
	TMatch, top, L2V		✓	0.5046	0.5139
Multi-source	OTG (cf. Chapter 6)	✓	✓	0.5856	0.6062
	CellAutoComplete (feat. I)	✓	✓	0.6641‡	0.6826‡
	CellAutoComplete (feat. I+II)	✓	✓	0.6844‡	0.7034‡
	CellAutoComplete (feat. I+II+III)	✓	✓	**0.7570**‡	**0.7641**‡

Source	Method	Sources used KB	Sources used TC	Empty included NDCG@5	Empty included NDCG@10
Single-source	KBLookupED	✓		0.2780	0.2806
	InfoGather, top, UNI		✓	0.4158‡	0.4302‡
	InfoGather, top, L2V		✓	0.4413	0.4537
	TMatch, top, UNI		✓	0.4297‡	0.4417‡
	TMatch, top, L2V		✓	0.4531	0.4624
Multi-source	OTG (cf. Chapter 6)	✓	✓	0.5185	0.5367
	CellAutoComplete (feat. I)	✓	✓	0.5766‡	0.5954‡
	CellAutoComplete (feat. I+II)	✓	✓	0.5905‡	0.6100‡
	CellAutoComplete (feat. I+II+III)	✓	✓	**0.6716**‡	**0.6785**‡

Eight platform.[5] For each cell, human assessors were presented with the page title (embedding the table), table caption, the core column entity, the heading column label, and a source document. The source document is either the DBpedia page of the core column entity or an existing table from the table corpus. Users were then asked to check if the missing cell value can be found within the source document, and, if yes, to provide the corresponding value (otherwise enter a designated special `Empty` value). Each instance was annotated by 7 assessors. The inter-annotator agreement in terms of Fleiss' kappa statistic was about 0.7 when using the knowledge base and 0.8 when using the table corpus as source. The former is considered as substantial, the latter is considered as almost perfect agreement [Landis and Koch, 1977]. The total expense of the crowdsourcing experiments was $770.

We then combine the correct values from these two sources as our ground truth. (We only use the KB and TC specific subsets in our analysis of specific sources in Sect. 7.5.2.) Table 7.7 provides a summary of our test collection. We find that, when using both sources, cells on average have over two pos-

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

sible correct values. It is further worth noting that the rate of empty cells is much lower when combining the two sources, attesting to their complementary nature.

7.4.5 Table Matching

To train our table matching models (InfoGather and TMatch in Sect. 7.2.1), we construct a training dataset. We group tables by topics and sample 50 tables with diverse topics (such as military, paleontology, sports, geography, etc.) from the corpus as input tables. Each table should have at least five rows and three columns. For each table, we utilize the query-based search methods in [Ahmadov et al., 2015a] to obtain a set of candidate tables. We ask 3 annotators to judge if the candidate table is highly relevant, relevant, or not relevant.

7.4.6 Evaluation Measures

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

7.5 Experimental Evaluation

This section presents evaluation results for the value finding task (Sect. 7.5.1) followed by further analysis of value sources (Sect. 7.5.2), features (Sect. 7.5.3), and specific examples (Sect. 7.5.4).

7.5.1 Evaluating Auto-Completion

We begin with the evaluation of the end-to-end cell value auto-completion task. Table 7.8 reports the results. At the top block of Table 7.8, we display the methods that use an individual source, either knowledge base (KB, line 1) or table corpus (TC, lines 2–5). These methods meant to serve as single-source baselines; they are further detailed in Sect. 7.5.2. The bottom block of Table 7.8 shows methods that utilize both sources. There is only one existing work in the literature that we find directly applicable: the On-the-Fly Table Generation (OTG) approach in Chapter 6. This method combines a knowledge base and a table corpus in a simple way, by always giving preference to the former source over the latter.

Looking at the results in Table 7.8, it is clear that the table corpus is a more effective source for value finding than the knowledge base. At the same time, they are complementary and combining the two yields substantial improvements. This is already witnessed for OTG, but to a much larger extent with our CellAutoComplete methods. Our best methods, using the complete feature set (cf. Table 7.6) outperforms OTG substantially, i.e., by over 26% on all evaluation metric and experimental conditions (lines 6 vs. 9). These improvements can be attributed to two main factors. First, instead of naively giving preference to the knowledge base over the table corpus, as in OTG, CellAutoComplete (feat. I) decides for each cell individually which source should be preferred, by considering the predicate-to-heading and heading-to-heading matching probabilities, among other signals. This makes a large difference, as can be observed in the scores (lines 6 vs. 7). Second, taking into account the semantic similarity of tables, when using a table corpus as source, makes a large difference. This is what feature group III contributes. We find that it brings in an over 10% relative improvement, see CellAutoComplete (feat. I+II) vs. (feat. I+II+III), i.e., the bottom two lines in Table 7.6). As for the second group of features, which aims at improving empty value prediction, we find that is has a small, but positive and significant impact (lines 7 vs. 8).

7.5.2 Analysis of Sources

Next, we analyze cell auto-completion performance using only a single source: a knowledge base (Table 7.9) and a table corpus (Table 7.10). As before, we distinguish between two settings, with `Empty` values excluded and included. We note that the ground truth is restricted to the specific source, therefore, it is different in the two cases (and also different from Table 7.8, which uses the union of the two).

Using a Knowledge Base

We compare two different KB-based value lookup methods, edit distance (ED) and matching probability (MP), in Table 7.9. The two methods yield virtually identical performance when `Empty` values are excluded. When `Empty` values are considered, ED performs significantly better than MP. Recall that our approach involves a γ threshold for `Empty` detection (cf. Eq. (7.3)). Here, we estimate this threshold using 5-fold cross-validation, and the average γ value is 0.8 for ED and 0.6 for MP. The reason that edit distance performs better is that it is more robust with respect to the value of γ. In other words, a single γ value performs well across different predicate-column heading pairs.

Using a Table Corpus

We consider (i) two table matching methods, InfoGather and TMatch;[6] (ii) two evidence combination strategies, top-ranked table (top) and all tables

[6]Additionally, we have also considered DTS [Nguyen et al., 2015] and the method in [Das Sarma et al., 2012] for table matching. However, both were inferior to InfoGather

Table 7.9: Value finding performance using a knowledge base. Significance of MP is tested against ED.

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

(all); and (iii) four heading label similarity methods, uniform (UNI), edit distance (ED), mapping probability (MP), and Label2Vec (L2V). Table 7.10 presents all possible combinations of these.

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

7.5.3 Feature Importance Analysis

In order to gain an understanding of which features contribute most to the effectiveness of our value ranking approach, we measure their importance in terms of Gini score. The results are shown in Fig. 7.6, ordered left to right from most to least important. Generally, features from group I and III are the most represented at the top ranks, while feature group II and table features dominate the bottom half of the ranking.

7.5.4 Cell-level Analysis

So far, we have reported on aggregate statistics. In our final experimental section, we perform an analysis on the level of individual cells. Recall that when creating the test collection, we have concealed the original cell values from the input tables, pretending that these were missing. In this part, we

in terms of effectiveness. Therefore, in the interest of space we only report only on InfoGather.

Table 7.10: Value finding performance using a table corpus. Highest score are boldfaced. Significance of TMatch (lines 9-16) is tested against InfoGather (lines 1-8).

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

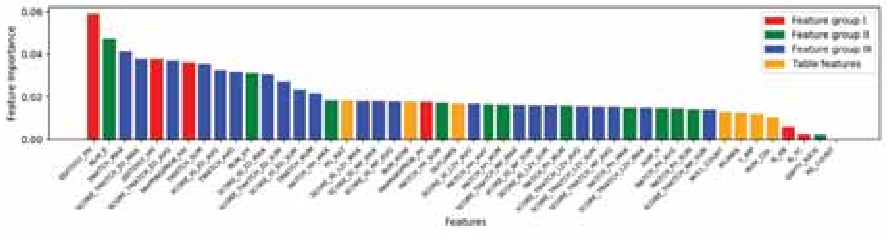


Figure 7.6: Feature importance measured in terms of Gini score.

compare these original cell values (referred to as *original*) with the values that were retrieved automatically by our approach (referred to as *found*).

Table 7.11 reports the overall statistics. The first and second lines of this table represent the cases where the cell was originally empty and had a value in the input table, respectively. The columns of the table correspond to how many different (valid) values were found by our approach. Below, we take a closer look at each of these cases, from top to bottom and from left to right.

- There are 20 cells, where originally the cell was empty and we also did

Table 7.11: Cell-level analysis, comparing the original (concealed) values in test tables against those found by our CellAutoComplete method.

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

not find a value (i.e., no difference).

- In 3 cases, we found the value for a cell that was originally empty. On such example is the "departure" time for "Hampton Roads" in a table about "Itinerary."
- In 5 cases, we identified two valid values for a cell that was originally empty. For instance, for the "type" column of "Polvorones", in a table about "Breads and pastries," both "shortbread" and "bread" are correct values.
- There are 154 cells where the cell is originally not empty, but we could not find its value. This is the category where our method failed. It turns out that in most of these cases, the given values exist only in the original tables (which were excluded from the corpus).
- In 205 cases, both *original* and *found* have the same single value (i.e., no difference).
- For 613 cells that are originally non-empty, we found multiple valid values. In many cases, *found* includes further values in addition to the original value. E.g., the original value is "Republican Party (United States)," while the found values also include "R" and "Republican." Another example is an athlete's "country," which is originally "Brazil at the Olympics", while the found values also include "Brazil." In some cases the granularity of the values differ, e.g., the "location" of "Pike" is "Levee Township, Pike County, Illinois" in the original table, while values in *found* also include "Hull, Illinois", "Detroit, Illinois", "Pittsfield, Illinois", and "Pearl, Illinoi." In other cases, there is no overlap between the values returned by *original* and *found*. There are several cases where the difference is in the value formats or in the granularity. E.g., the original table contains "1982" as the "death" date of "Hugh John Flemming," while the value we returned from the knowledge base is "1982-10-16." Another reason for the differences has to do with temporal mismatch, i.e., one of the sources is out-of-date.Finally, there are also some genuine cases of conflicting values. E.g., the "open date" of "Kannon Station" is "1923-07-05" in the original table, while in DBpedia the "opening year" is "1913-01-01." Similarly, the "Platform" of "Okular" is "MS" according to one Wikipedia table, while it

is "Unix-like" in DBpedia.

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

7.6 Summary and Conclusions

We have addressed the task of finding cell values, given an input relational table, by developing the CellValueFinder framework. It consists of preprocessing, candidate finding, and value ranking steps, and utilizes a knowledge base and a table corpus as data sources. The main innovative elements of our approach include (i) dealing with multiple, possibly conflicting values, (ii) supplementing the identified values with supporting evidence, (iii) combining evidence across multiple sources, (iv) considering multiple value types, and (v) identifying cases where a table cell should be left empty. We have demonstrated the effectiveness of our method on a purpose-built test collection and have advanced the current state-of-the-art by a considerable margin.

As the last task of this thesis, the CellValueFinder framework complements the tasks of table completion (Chapter 5) and table generation (Chapter 6). We will detail possible future directions for this task in Chapter 8.

Chapter 8

Future Directions and Conclusions

In this chapter, we will summarize our main contributions and then discuss the future directions of each task proposed in this thesis.

8.1 Main Contributions

The main motivation for this thesis was to facilitate tables with smart capabilities. To address it, we proposed a set of tasks such as searching, generating and completing tables.

Table search. We proposed two table search tasks based on two types of queries. Keyword table search addresses the ad hoc table retrieval in order to reply a keyword query with a ranked list of tables. Query-by-table deals with the scenario when the query is a table. We developed semantic matching frameworks for both tasks, where queries and tables can be represented using semantic concepts (bag-of-entities and bag-of-words) as well as continuous dense vectors (word and graph embeddings) in a uniform manner. We conducted multiple similarity measures for matching those semantic representations. Apart from the query type, the main difference between the two tasks is that we additionally incorporate the similarity between elements that are of different types (cross-element matching) in the later task. We further developed two purpose-built test collections based on Wikipedia tables and demonstrated superiority over a number of baselines.

Table completion. For table completion, we introduced the idea of assisting users with completing tables by recommending additional entities and additional column headings to be added to the table. Correspondingly, we focused on two specific tasks: row population and column population. Taking the seed table as input, we proposed generative probabilistic methods for both tasks. The experimental evaluation simulates different stages of the user entering content into an actual table. We also detailed that our methods outperform existing methods from the literature by assembling the complementary components.

Table Generation. We proposed the task of on-the-fly table generation that automatically compiles a table in response to a query. We addressed the task by composing it into three specific subtasks: core column entity ranking, schema determination and value lookup. We employed a feature-based approach for entity ranking and schema determination, combining deep semantic features with task-specific signals. We assumed that core column entity ranking and schema determination can reinforce each other, and verified it by implementing the idea of iterative refinement. We combine information from existing tables and a knowledge base for value lookup. We evaluated our methods using two sets of entity-oriented queries.

Table value finding. We introduced the task of finding table cell values with supporting evidence, to support users in the labor-intensive process of creating relational tables. To address it, we proposed the CellValueFinder framework, which consists of preprocessing, candidate value finding, and value ranking. We designed a feature-based method by combining the signals from the tasks of heading-to-heading and heading-to-predicate matching components. We show that our LTR method outperforms existing data augmentation techniques.

8.2 Future Directions

A large number of recently published studies, which are concerned with problems such as table summarization, table generation, and supporting decision-making, suggest that tables will remain a key research area for years to come. In the following, we will discuss the future directions related to our search: table search, table completion, table generation, table value finding and the choice of table corpus.

Table search. Our semantic matching frameworks designed for table search ignore the tabular data from the non-core columns. As such, it would be interesting to investigate the way to utilize them. We are further interested in evaluating the utility of our approach with user studies in a task-based scenario. Additionally, we wish to relax the requirements regarding the focus on Wikipedia relational tables, and make our methods applicable to other types of tables, like scientific tables [Gao and Callan, 2017] or Web tables. This task is one of the core tasks that was started in the early days and remains to be an active research topic ever since. One topic that deserves attention in our opinion, but has not been explored yet, is the presentation of table search results. For example, for large tables, how should appropriate snippets (summaries) be generated for search result pages?

Table completion. With the huge success of Learn-To-Rank methods and recent success of Deep Learning method in IR community, there is an increasing demand for investigate the methods to assist to complete tables in a supervised manner. Similar to table search, table completion in this thesis limits itself from the constraints such as relying on a relational seed table.

Table Generation. We designed an iterative method for the core column entity ranking and schema determination. However, we relied on the posterior evaluations instead of finding a priori termination condition. As a result, a study on the proper termination condition is needed here. Alternatively, a more efficient method for generating tables will enhance the efficiency of the current iterative method. Besides, on-the-fly table generation aims to answer the queries that target at a number of entities. In a real table generation system, a module to distinguish the query type is needed. As such, it will figure out when a relational table is needed as the answer.

Table value finding. There is a solid body of work on augmenting existing tables with additional data, extracted either from other tables or from knowledge bases. However, there are at least two issues that remain. One is tapping into the large volumes of unstructured sources (e.g., web pages). The other is combining data from multiple sources, which brings about a need for techniques to deal with conflicting information. Corresponding to above concerns, we developed methods for finding values from structured sources in this thesis. Additionally, we are interested in incorporating evidence from unstructured text, e.g., web pages. Finally, we wish to explicitly address the temporal aspects of certain entity attributes. While our approach has been developed with a specific application in mind, the core techniques we developed for table value finding may also be put to use in other problem contexts, such as information extraction, populating KBs from tables, and truth/fact finding. We see several avenues for future work.

Table corpus. Table tasks in this thesis are conducted under a few assumptions such as Wikipedia relational tables. In the future, we plan to test our methods on a more heterogeneous collections of tables form the Web, which vary more quality-wise than Wikipedia tables.

Other tasks. The abundant information embedded in tables can be utilized in many other tasks like question answering, knowledge base augmentation, and so on. There is a body of work on extracting facts from the tables. The facts extracted from tables can be used as answers for natural language questions. They additionally can also enrich the current knowledge bases.

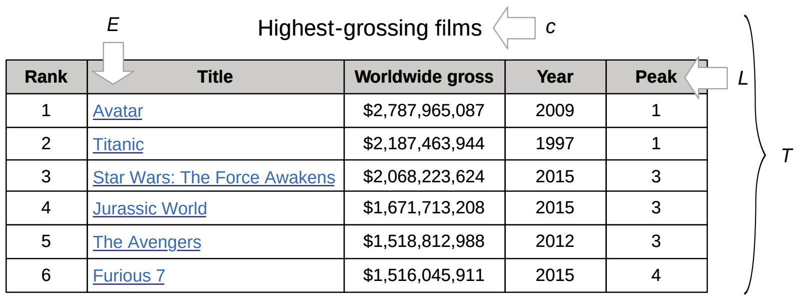
Appendix A

SmartTable: A Spreadsheet Program with Intelligent Assistance

Tables can be found in vast quantities on the Web, and spreadsheet programs are among the most commonly used desktop applications. Our objective is to equip spreadsheet programs with intelligence assistance capabilities, to aid users while working with tables. To demonstrate the techniques we proposed in Chapter 5 of this thesis, we introduce SmartTable, an online spreadsheet tool equipped with smart assistance capabilities. As before, we work with relational tables; See Figure A.1 for an illustration.

There exists a number of online tools and resources for table-related tasks, such as table search (Google Fusion Tables [Cafarella et al., 2009] and WikiTables [Bhagavatula et al., 2015]), question answering [Pasupat and Liang, 2015b], and entity linking in tables [Bhagavatula et al., 2015]. To the best of our knowledge, our system, called *SmartTable*, is the first online spreadsheet program that provides intelligent table content recommendation. Specifically, our application is capable of providing two kinds of assistance: (i) recommending additional entities, from an underlying knowledge base, to be added to the core column (row population) and (ii) recommending additional entity attributes to be included as columns (column population). Such recommendations are particularly useful in scenarios with an exploratory or recall-oriented nature, i.e., when the user does not have a very clear idea beforehand as to what should be included in the table. Additionally, SmartTable also provides regular table operations, such as adding, deleting, and moving rows and columns, editing cells, and supporting various value types (entities, numbers, currencies, dates, etc.).

Both types of assistance, that is, row and column population, are based on probabilistic models that we developed in Chapter 5. The main contributions of this work are twofold. First, we integrate the above assistance functionality into an online spreadsheet application. Second, we describe the



Highest-grossing films

Rank	Title	Worldwide gross	Year	Peak
1	Avatar	$2,787,965,087	2009	1
2	Titanic	$2,187,463,944	1997	1
3	Star Wars: The Force Awakens	$2,068,223,624	2015	3
4	Jurassic World	$1,671,713,208	2015	3
5	The Avengers	$1,518,812,988	2012	3
6	Furious 7	$1,516,045,911	2015	4

Figure A.1: Example of a relational table T, where c is the table caption, E denotes the core column entities $E = \{e_1, \ldots, e_n\}$, and L is the set of column labels $L = \{l_1, \ldots, l_m\}$.

task-specific indexing structures employed, and evaluate the efficiency of our implementation in terms of response time. SmartTable is implemented using a HTML5 front-end and a Python+ElasticSearch back-end. It uses DBpedia as the underlying knowledge base and a corpus of 1.6M tables extracted from Wikipedia. The implementation is released as open source. The application is available at `http://smarttable.cc`.

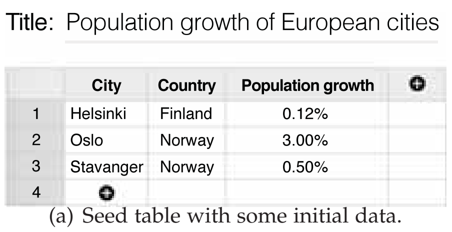
A.1 Overview

In this section, we provide an overview of the functionality of the SmartTable application, by walking through the process of creating a table from scratch.

- Initially, we start with an empty table, with the table caption, core column entities, column labels, and cell values waiting to be filled. The user is expected to add a few entities and column labels first, along with an optional table caption, to supply the system with some data to base recommendations on. We shall refer to this (incomplete) table as the *seed table*. See Fig. A.2(a).

- When adding entities to the core column, the user is presented with a ranked list of suggestions. Additionally, the user can search the underlying knowledge base for entities. See Fig. A.2(b).

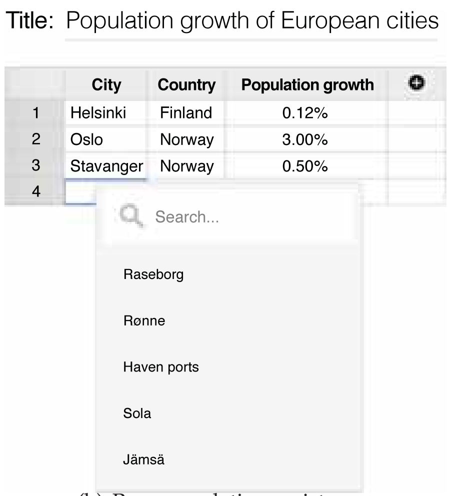
- When adding new columns, the user needs to specify the data type for that column (which can be one of entity, text, date, number, currency, or percentage) and provide a label for that column. For the latter, a ranked list of suggestions are offered, along with a search box to search for additional labels. See Fig. A.2(c).



Title: Population growth of European cities

	City	Country	Population growth	⊕
1	Helsinki	Finland	0.12%	
2	Oslo	Norway	3.00%	
3	Stavanger	Norway	0.50%	
4	⊕			

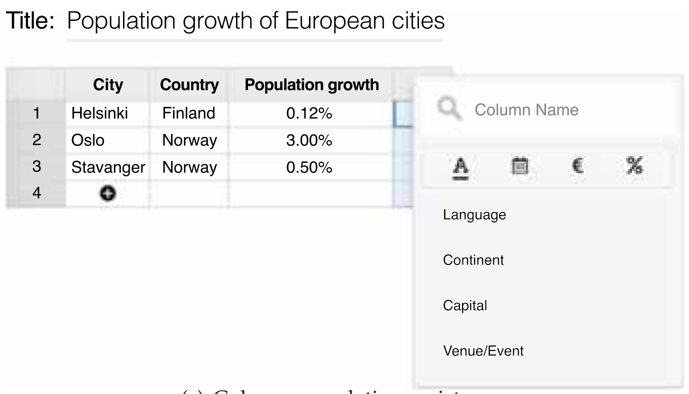
(a) Seed table with some initial data.



Title: Population growth of European cities

	City	Country	Population growth	⊕
1	Helsinki	Finland	0.12%	
2	Oslo	Norway	3.00%	
3	Stavanger	Norway	0.50%	
4				

(b) Row population assistance.



Title: Population growth of European cities

	City	Country	Population growth
1	Helsinki	Finland	0.12%
2	Oslo	Norway	3.00%
3	Stavanger	Norway	0.50%
4	⊕		

(c) Column population assistance.

Figure A.2: Screenshots from the SmartTable system.

A.2 Methods

In this section, we introduce the methods underlying the assistance functionality. We refer to Fig. A.1 for the notation used for the various table elements. As for the data, we employ a table corpus (TC) extracted from Wikipedia and use DBpedia as the knowledge base (KB); further details about the datasets are given in Sect. 5.4.1.

A.2.1 Row population

Row population is the task of generating a ranked list of entities to be added to the core column of a given seed relational table. The row population task is split into two sub-tasks, which are candidate selection and ranking entities, respectively.

Candidate selection

We identify candidate entities using both the knowledge base and the table corpus. From the knowledge base, we take entities that share the assigned semantic categories with those of the seed entities. From the table corpus, we first find tables similar to the seed table, based on table caption, core column entities, and column heading labels. Then, we take the core column entities from those similar tables as candidates.

Ranking entities

For efficiency, we implement a variant probabilistic model based on that proposed in Chapter 5, which is a multi-conditional probability:

$$P(e|E, L, c) \propto P(e|E)P(L|e)P(c|e) ,$$

where $P(e|E)$ is entity similarity, $P(L|e)$ denotes column labels likelihood, and $P(c|e)$ is caption likelihood. We refer to Section 5.2.2 for the estimation of these components. We only differ in the estimation of $P(L|e)$. In our original approach, in Chapter 5, this estimate was a two-component mixture. Due to efficiency considerations, we use a simplified version here. The relative difference in terms of effectiveness is below 5%. Specifically, we set:

$$P(L|e) = \sum_{l \in L} \prod_{t \in l} \frac{tf(t, e) + \mu P(t|\theta)}{|e| + \mu} ,$$

where $tf(t, e)$ is the term frequency of t in the column labels of tables containing e and $|e|$ is the sum of all term frequencies for e. The collection language model $P(t|\theta)$ is computed based on the column labels of all tables in TC.

A.2.2 Column population

Column population is the task of generating a ranked list of column labels to be added to the column headings of a given seed table. It is also implemented as a sequence of two steps: candidate selection and column label ranking.

Candidate selection

Candidate labels are obtained from related tables. To find related tables, we use (i) the table caption, (ii) table entities, and (iii) seed column heading labels as queries. From the matching tables, column labels are extracted as candidates.

Ranking column labels

The related tables, identified in the candidate selection stage, are also utilized in the ranking step. According to the model in Chapter 5, the probability of a candidate column label is given by:

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

where T represents a related table, $P(l|T)$ is the label's likelihood given T, and $P(T|E,c,L)$ expresses that table's relevance. We refer to Section. 5.3.2 for the estimation of these probabilities.

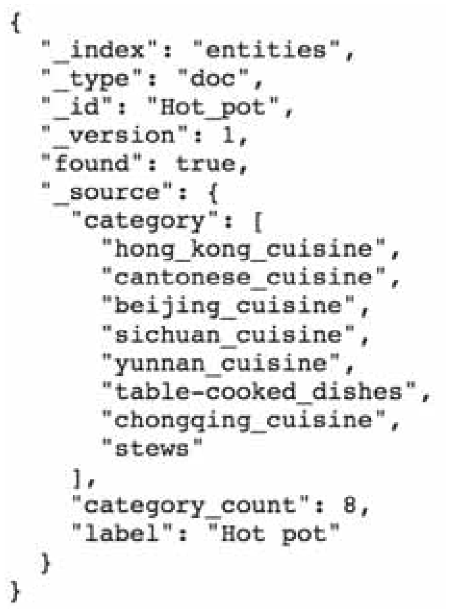
A.3 Implementation

In this section, we describe the datasets used and indices built, along with technical details of our implementation.

A.3.1 Datasets

We rely on two data sources: a table corpus and a knowledge base. The knowledge base is DBpedia, version 2015-10.[1] We filter out entities that do not have a short textual description (abstract). After filtering, we are left with a total of 4.6M entities. As for the table corpus, we use the WikiTables collection [Bhagavatula et al., 2015], which comprises of 1.65M tables, extracted from Wikipedia. We preprocess tables as follows. Entities are marked up in the original table with hyperlinks. If the link points to an entity that exists in DBpedia, we replace that link with the corresponding entity identifier. Otherwise, we replace the link with the anchor text.

[1]http://wiki.dbpedia.org/dbpedia-dataset-version-2015-10



```
{
  "_index": "entities",
  "_type": "doc",
  "_id": "Hot_pot",
  "_version": 1,
  "found": true,
  "_source": {
    "category": [
      "hong_kong_cuisine",
      "cantonese_cuisine",
      "beijing_cuisine",
      "sichuan_cuisine",
      "yunnan_cuisine",
      "table-cooked_dishes",
      "chongqing_cuisine",
      "stews"
    ],
    "category_count": 8,
    "label": "Hot pot"
  }
}
```

Figure A.3: Example entry from the entity index.

A.3.2 Indices

We build the following inverted indices:

Table index It contains 1.65M Wikipedia tables (6.4GB). For each table, the following fields are stored: page title, section title, table caption, column labels, table data, and core column entities.

Entities It contains 4.6M DBpedia entities (2GB). For each entity, we store its canonical name (label), and the list and number of categories it is assigned to. See Fig. A.3 for an example.

Categories We use Wikipedia's category system, comprising of around 1M categories. For each category, we store the list of entities that are assigned to that category. This index occupies 2GB.

A.3.3 Implementation

SmartTable is a web application that is comprised of a HTML5 front-end and a back-end based on Python and Elasticsearch.

Front-end

The front-end stack is made up of HTML, CSS, and JavaScript (ECMAScript6 standard). We build on a third-party JavaScript spreadsheet framework

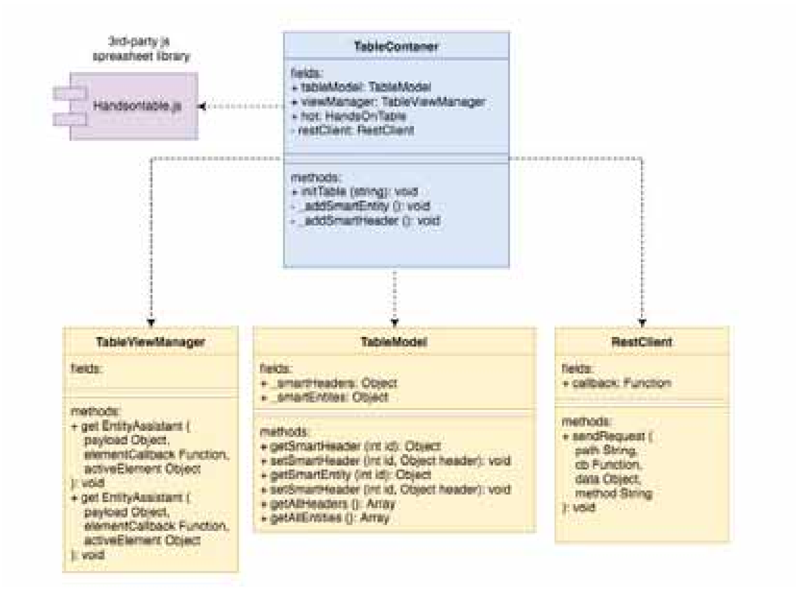

3rd-party js spreasheet library
Handsontable.js
TableContaner
fields:
+ tableModel: TableModel
+ viewManager: TableViewManager
+ hot: HandsOnTable
- restClient: RestClient
methods:
+ initTable (string): void
- _addSmartEntity (): void
- _addSmartHeader (): void
TableViewManager
fields:
methods:
+ get EntityAssistant (payload Object, elementCallback Function, activeElement Object): void
+ get EntityAssistant (payload Object, elementCallback Function, activeElement Object): void
TableModel
fields:
+ _smartHeaders: Object
+ _smartEntities: Object
methods:
+ getSmartHeader (int id): Object
+ setSmartHeader (int id, Object header): void
+ getSmartEntity (int id): Object
+ setSmartHeader (int id, Object header): void
+ getAllHeaders (): Array
+ getAllEntities (): Array
RestClient
fields:
+ callback: Function
methods:
+ sendRequest (path String, cb Function, data Object, method String): void

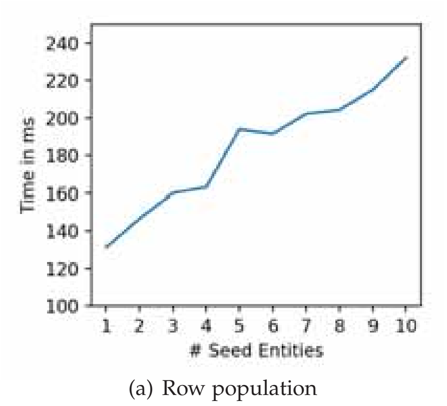
Figure A.4: Overview of the application front-end.

called Handsontable,[2] which provides a rich set of functionality for tables, including sorting, conditional formatting, contextual menus, moveable and resizable rows and column, etc. Additionally, we utilize the `Gulp.js`, `Babel.js`, and `Node.js` JavaScript libraries.

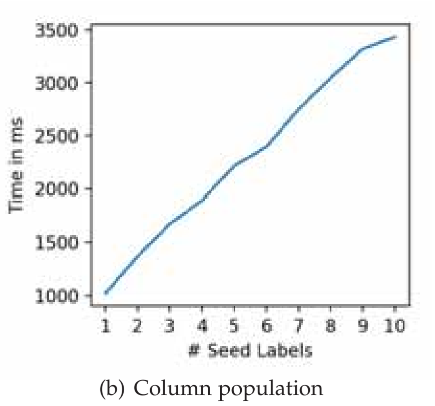
For development, we follow the MVC (Model View Controller) software architecture pattern. The system is divided into self-contained components that are easy to debug and maintain, with loose coupling and modularity between the fundamental parts. Figure A.4 provides an overview. At the center of front-end lies the TableContainer class, connecting the following components:

- Handontable.js: Third party JavaScript spreadsheet framework.
- TableViewManager.js: Smart Assistant view controller.
- TableModel.js: Provides storage and accessibility to all core column entities and column heading labels.
- RestClient.js: Communication component, which is responsible for request sending and response provision via the respective callback calls.

[2]https://handsontable.com/



(a) Row population



(b) Column population

Figure A.5: Performance in terms of response time.

Back-end

The back-end consists of two parts: a web server and a recommendation engine. The main role of the former is to connect the front-end spreadsheet application (client) with the recommendation engine. The web server is implemented in Python, using the Flask framework.[3] Communication is done over HTTP, with request and response messages encoded in JSON format. The recommendation engine is responsible for generating the ranked list of suggestions (entities and column labels). It uses Elasticsearch as the underlying indexing and retrieval engine. All indices are built using the Nordlys toolkit [Hasibi et al., 2017a].[4]

A.4 Evaluation

In Chapter 5, we have performed an extensive evaluation of the row and column population methods in terms of effectiveness. Here, we evaluate our system in terms of efficiency. We measure response time as the time elapsed between receiving the request and sending off the response on the back-end, i.e., net computation time without the network overhead. Using 10 random tables, we vary the number of core column entities (seed entities) and the number of heading column labels (seed labels). The measurements are repeated 10 times and averages are reported in Figs. A.5(a) and A.5(b). We can observe that, in both cases, response time grows linearly with the size of the input. For row population, the response time is beyond 250ms, even with the largest input size, which is considered very acceptable. For

[3]http://flask.pocoo.org/
[4]http://nordlys.cc

column population, responses are a magnitude slower. This is due to the fact that we consider all related tables in our scoring formula. Limiting the computations to the top-*k* most similar tables may provide a solution; it is left for future work to find a *k* value that provides a good trade-off between effectiveness and efficiency.

A.5 Summary and Conclusions

We have introduced SmartTable, an online spreadsheet application that is equipped with smart assistance capabilities. Specifically, we aid users working with relational tables by suggesting them additional entities and column heading labels to be included in the table. In future work, we consider diversifying recommendations and plan to extend the scope of content recommendation to data cells as well, by suggesting possible values for them. Furthermore, we intend to integrate table search and table generation functionality, which we developed in this thesis.

Appendix B

Resources Developed in this Thesis

To ensure the reproducibility of our research, we made publicly available a collection of resources developed in this thesis, which are listed in Table B.1. $\mathcal{A}$ denotes appendix.

Table B.1: Resources developed in this thesis.

Chapter	Description	Link
3	Test collection, feature file, and run files related to keyword table search	https://github.com/iai-group/www2018-table
5	Code and test collections related to table completion	https://github.com/iai-group/sigir2017-table
6	Test collections, feature files, and run files related to table generation	https://github.com/iai-group/sigir2018-table
7	Test collections, feature files, and run files related to table cell completion	https://github.com/iai-group/cikm2019-table
$\mathcal{A}$ A	Code of SmartTable demo	https://github.com/iai-group/SmartTable http://smarttable.cc/

Bibliography

A. Ahmadov, M. Thiele, J. Eberius, W. Lehner, and R. Wrembel. Towards a hybrid imputation approach using web tables. In *Proc. of BDC '15*, pages 21–30, 2015a.

A. Ahmadov, M. Thiele, J. Eberius, W. Lehner, and R. Wrembel. Towards a hybrid imputation approach using web tables. In *Proc. of BDC '15*, pages 21–30, 2015b.

M. Anan and G. Avigdor. On the stable marriage of maximum weight royal couples. In *Proc. of IIweb '07*, pages 1–6, 2007.

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *CoRR*, cmp-lg/9503016, 1995.

S. Balakrishnan, A. Y. Halevy, B. Harb, H. Lee, J. Madhavan, A. Rostamizadeh, W. Shen, K. Wilder, F. Wu, and C. Yu. Applying webtables in practice. In *Proc. of CIDR '15*, 2015.

K. Balog, M. Bron, and M. De Rijke. Query modeling for entity search based on terms, categories, and examples. *ACM Trans. Inf. Syst.*, 29(4):22:1–22:31, Dec. 2011.

S. Banerjee, S. Chakrabarti, and G. Ramakrishnan. Learning to rank for quantity consensus queries. In *Proc. of SIGIR '09*, pages 243–250, 2009.

J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proc. of EMNLP '13*, pages 1533–1544, 2013.

C. S. Bhagavatula, T. Noraset, and D. Downey. Methods for exploring and mining tables on wikipedia. In *Proc. of IDEA '13*, pages 18–26, 2013.

C. S. Bhagavatula, T. Noraset, and D. Downey. Tabel: Entity linking in web tables. In *Proc. of ISWC 2015*, pages 425–441, 2015.

K. Braunschweig, M. Thiele, J. Eberius, and W. Lehner. Column-specific context extraction for web tables. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 1072–1077, 2015a. ISBN 978-1-4503-3196-8.

K. Braunschweig, M. Thiele, and W. Lehner. From web tables to concepts: A semantic normalization approach. In *Conceptual Modeling*, IC3K '16, pages 247–260, 2015b.

K. Braunschweig, M. Thiele, E. Koci, and W. Lehner. Putting web tables into context. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016)*, IC3K '16, pages 158–165, 2016.

M. Bron, K. Balog, and M. de Rijke. Example based entity search in the web of data. In *Proc. of ECIR'13*, pages 392–403, 2013.

M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *Proc. VLDB Endow.*, 1:538–549, 2008a.

M. J. Cafarella, A. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *Proc. of WebDB '08*, 2008b.

M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *Proc. of VLDB Endow.*, 2:1090–1101, 2009.

J. Chen, C. Xiong, and J. Callan. An empirical study of learning to rank for entity search. In *Proc. of SIGIR '16*, pages 737–740, 2016.

Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search Over the Web*, SS@ '13, pages 1–8, New York, NY, USA, 2013. ACM.

F. Chirigati, J. Liu, F. Korn, Y. W. Wu, C. Yu, and H. Zhang. Knowledge exploration using tables on the web. *Proc. of VLDB Endow.*, 10:193–204, 2016.

E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

E. Crestan and P. Pantel. Web-scale table census and classification. In *Proc. of WSDM '11*, pages 545–554, 2011.

A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *Proc. of SIGMOD '12*, pages 817–828, 2012.

L. Deng, S. Zhang, and K. Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proc. of SIGIR '19*, 2019.

X. Dong and A. Y. Halevy. Malleable schemas: A preliminary report. In *Proc. of the WebDB '05*, 2005.

X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proc. of KDD '14*, pages 601–610, 2014.

J. Eberius, K. Braunschweig, M. Hentsch, M. Thiele, A. Ahmadov, and W. Lehner. Building the dresden web table corpus: A classification approach. In *Proc. of BDC '15*, pages 41–50, 2015.

V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides. Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In *Proc. of ISWC '17*, pages 260–277. Springer, 2017.

A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In *Proc. of KDD '14*, pages 1156–1165, 2014.

J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *Proc. of ICDE '14*.

Y. Fan, L. Pang, J. Hou, J. Guo, Y. Lan, and X. Cheng. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270*, 2017.

J. Fleiss et al. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76:378–382, 1971.

D. Ganguly, D. Roy, M. Mitra, and G. J. Jones. Word embedding based generalized language model for information retrieval. In *Proc. of SIGIR '15*, pages 795–798, 2015.

K. Y. Gao and J. Callan. Scientific table search using keyword queries. *CoRR*, abs/1707.03423, 2017.

V. Govindaraju, C. Zhang, and C. Ré. Understanding tables in context using standard nlp toolkits. In *ACL*, 2013.

M. Grbovic, N. Djuric, V. Radosavljevic, F. Silvestri, and N. Bhamidipati. Context- and content-aware embeddings for query rewriting in sponsored search. In *Proc. of SIGIR '15*, pages 383–392, 2015.

J. Guo, Y. Fan, Q. Ai, and W. B. Croft. A deep relevance matching model for ad-hoc retrieval. In *Proc. of CIKM '16*, pages 55–64, 2016.

R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *VLDB Endow.*, 2(1):289–300, Aug. 2009. ISSN 2150-8097.

F. Hasibi, K. Balog, D. Garigliotti, and S. Zhang. Nordlys: A toolkit for entity-oriented and semantic search. In *Proceedings of SIGIR '17*, pages 1289–1292, 2017a.

F. Hasibi, F. Nikolaev, C. Xiong, K. Balog, S. E. Bratsberg, A. Kotov, and J. Callan. Dbpedia-entity v2: A test collection for entity search. In *Proc. of SIGIR '17*, pages 1265–1268, 2017b.

O. Hassanzadeh, M. J. Ward, M. Rodriguez-Muro, and K. Srinivas. Understanding a large corpus of web tables through matching with knowledge bases: an empirical study. volume 1545 of *CEUR Workshop Proceedings*, pages 25–34. CEUR-WS.org, 2015.

Y. He and D. Xin. Seisa: set expansion by iterative similarity aggregation. In *Proc. of WWW '11*, pages 427–436, 2011.

Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-data-by-example (tde): An extensible search engine for data transformations. *Proc. VLDB Endow.*, 11(10):1165–1177, June 2018.

V. Hung, B. Benatallah, and R. Saint-Paul. Spreadsheet-based complex data transformation. In *Proc. of CIKM '11*, pages 1749–1754, 2011.

Y. Ibrahim, M. Riedewald, and G. Weikum. Making sense of entities and quantities in web tables. In *Proc. of CIKM '16*, pages 1703–1712, 2016. ISBN 978-1-4503-4073-1.

T. Kenter and M. de Rijke. Short text similarity with word embeddings. In *Proc. of CIKM '15*, pages 1411–1420, 2015.

D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*, 2014. http://arxiv.org/abs/1412.6980.

A. Kopliku, M. Boughanem, and K. Pinel-Sauvagnat. Towards a framework for attribute retrieval. In *Proc. of CIKM '11*, pages 515–524, 2011.

J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33, 1977.

L. R. Lautert, M. M. Scheidt, and C. F. Dorneles. Web table taxonomy and formalization. *SIGMOD Rec.*, 42(3):28–33, Oct. 2013. ISSN 0163-5808.

O. Lehmberg and C. Bizer. Web table column categorisation and profiling. In *Proc. of WebDB '16*, pages 4:1–4:7, 2016.

O. Lehmberg and C. Bizer. Stitching web tables for improving matching quality. *Proc. VLDB Endow.*, 10(11):1502–1513, Aug. 2017.

O. Lehmberg, D. Ritze, P. Ristoski, R. Meusel, H. Paulheim, and C. Bizer. The mannheim search join engine. *Web Semant.*, 35:159–166, 2015.

O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. A large public corpus of web tables containing time and context metadata. In *Proc. of WWW '16 Companion*, pages 75–76, 2016.

F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, Sept. 2014. ISSN 2150-8097.

Y. Li, H. Yang, and H. V. Jagadish. Nalix: An interactive natural language interface for querying xml. In *Proc. of SIGMOD '05*, pages 900–902, 2005.

G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. of VLDB Endow.*, 3: 1338–1347, 2010.

T.-Y. Liu. *Learning to Rank for Information Retrieval*. Springer Berlin Heidelberg, 2011.

C. Macdonald, R. L. T. Santos, and I. Ounis. On the usefulness of query features for learning to rank. In *Proc. of CIKM '12*, pages 2559–2562, 2012.

J. Manotumruksa, C. MacDonald, and I. Ounis. Modelling user preferences using word embeddings for context-aware venue recommendation. *CoRR*, abs/1606.07828, 2016.

S. Metzger, R. Schenkel, and M. Sydow. Qbees: query by entity examples. In *Proc. of CIKM '13*, pages 1829–1832, 2013.

S. Metzger, R. Schenkel, and M. Sydow. Aspect-based similar entity search in semantic knowledge graphs with diversity-awareness and relaxation. In *Proc. of WI-IAT '14*, pages 60–69, 2014.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS '13*, pages 3111–3119, 2013.

D. Milne and I. H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *In Proc. of AAAI 2008*, 2008.

B. Mitra, E. T. Nalisnick, N. Craswell, and R. Caruana. A dual embedding space model for document ranking. *CoRR*, abs/1602.01137, 2016.

E. Muñoz, A. Hogan, and A. Mileo. Using linked data to mine rdf from wikipedia's tables. In *Proc. of WSDM '14*, pages 533–542, 2014.

V. Mulwad, T. Finin, Z. Syed, and A. Joshi. Using linked data to interpret tables. In *Proc. of COLD'10*, pages 109–120, 2010.

V. Mulwad, T. Finin, and A. Joshi. Semantic message passing for generating linked data from tables. In *Proc. of ISWC 2013*, pages 363–378, 2013.

F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *Proc. VLDB Endow.*, 11(7):813–825, Mar. 2018. ISSN 2150-8097.

A. Neelakantan, Q. V. Le, and I. Sutskever. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834, 2015.

T. T. Nguyen, Q. V. H. Nguyen, W. Matthias, and A. Karl. Result selection and summarization for web table search. In *ISDE '15*, pages 231–242, 2015.

K. Nishida, K. Sadamitsu, R. Higashinaka, and Y. Matsuo. Understanding the semantic structures of tables with a hybrid deep neural network architecture. In *Proc. of AAAI*, pages 168–174, 2017.

P. Ogilvie and J. Callan. Combining document representations for known-item search. In *Proc. of SIGIR '03*, pages 143–150, 2003.

P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas. Web-scale distributional similarity and entity set expansion. In *Proc. of EMNLP '09*, pages 938–947, 2009.

P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *Proc. of ACL '15*, pages 1470–1480, 2015a.

P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *Proc. of ACL '15*, pages 1470–1480, 2015b.

P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *Proc. of ACL '15*, pages 1470–1480, 2015c.

J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *Proc. of EMNLP '14*, pages 1532–1543, 2014.

B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proc. of KDD '14*, pages 701–710, 2014.

R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *Proc. of VLDB Endow.*, 5:908–919, 2012.

A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proc. of IUI '03*, pages 149–157, 2003.

T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 13(4): 346–374, Aug 2010.

H. Raviv, O. Kurland, and D. Carmel. Document retrieval using entity-based language models. In *Proc. of SIGIR '16*, pages 65–74, 2016.

P. Ristoski and H. Paulheim. RDF2vec: RDF graph embeddings for data mining. In *Proc. of ISWC '16*, pages 498–514, 2016.

D. Ritze and C. Bizer. Matching web tables to dbpedia - A feature utility study. In *Proc. of EDBT '17*, pages 210–221, 2017.

D. Ritze, O. Lehmberg, and C. Bizer. Matching html tables to dbpedia. In *Proc. of WIMS '15*, pages 10:1–10:6, 2015.

D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proc. of WWW '16*, pages 251–261, 2016.

P. Saleiro, N. Milic-Frayling, E. Mendes Rodrigues, and C. Soares. Relink: A research framework and test collection for entity-relationship retrieval. In *Proc. of SIGIR '17*, pages 1273–1276, 2017.

S. Sarawagi and S. Chakrabarti. Open-domain quantity queries on web tables: Annotation, response, and consensus models. In *Proc. of KDD '14*, pages 711–720, 2014.

Y. A. Sekhavat, F. D. Paolo, D. Barbosa, and P. Merialdo. Knowledge base augmentation using tabular data. In *Proceedings of WWW'14)*, 2014a.

Y. A. Sekhavat, F. D. Paolo, D. Barbosa, and P. Merialdo. Knowledge base augmentation using tabular data. In *Proc. of CEUR '15*, 2014b.

W. Shen, J. Wang, and J. Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Trans. Knowl. Data Eng.*, 27(2):443–460, feb 2015.

Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 373–374, 2014. ISBN 978-1-4503-2745-9.

C. G. M. Snoek, M. Worring, and A. W. M. Smeulders. Early versus late fusion in semantic video analysis. In *Proc. of MULTIMEDIA '05*, pages 399–402, 2005.

H. Sun, H. Ma, X. He, W.-t. Yih, Y. Su, and X. Yan. Table cell search for question answering. In *Proc. of WWW '16*, pages 771–782, 2016.

Z. S. Syed. *Wikitology: A Novel Hybrid Knowledge Base Derived from Wikipedia*. PhD thesis, Catonsville, MD, USA, 2010. AAI3422868.

J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proc. of WWW '15*, pages 1067–1077, 2015.

S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin. Parallel boosted regression trees for web search ranking. In *Proc. of WWW '11*, pages 387–396, 2011.

P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, 4: 528–538, 2011.

I. Vulić and M.-F. Moens. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proc. of SIGIR '15*, pages 363–372, 2015.

C. Wang, K. Chakrabarti, Y. He, K. Ganjam, Z. Chen, and P. A. Bernstein. Concept expansion using web tables. In *Proc. of WWW '15*, pages 1198–1208, 2015a.

H. Wang, A. Liu, J. Wang, B. D. Ziebart, C. T. Yu, and W. Shen. Context retrieval for web tables. In *Proc. of ICTIR '15*, pages 251–260, 2015b.

J. Wang, G. Li, and J. Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proc. of ICDE '11*, pages 458–469, 2011.

J. Wang, H. Wang, Z. Wang, and K. Q. Zhu. Understanding tables on the web. In *Proceedings of ER'12*, pages 141–155, 2012.

J. Wang, G. Li, and J. Feng. Extending string similarity join to tolerant fuzzy token matching. *ACM Trans. Database Syst.*, 39(1):1–45, 2014.

R. C. Wang and W. W. Cohen. Iterative set expansion of named entities using the web. In *Proc. of ICDM '08*, pages 1091–1096, 2008.

Y. Wang and Y. He. Synthesizing mapping relationships using table corpus. In *Proc. of SIGMOD '17*, pages 1117–1132, 2017.

Y. Wang and J. Hu. Detecting tables in html documents. In *Proc. of DAS '02*, pages 249–260, 2002a.

Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proc. of WWW '02*, pages 242–250, 2002b.

T. Wu, S. Yan, Z. Piao, L. Xu, R. Wang, and G. Qi. Entity linking in web tables with multiple linked knowledge bases. In *Semantic Technology*, pages 239–253. Springer International Publishing, 2016.

C. Xiong, J. Callan, and T.-Y. Liu. Word-entity duet representations for document ranking. In *Proc. of SIGIR '17*, pages 763–772, 2017.

M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the Web of Data. In *Proc. of EMNLP-CoNLL '12*, pages 379–390, 2012.

M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *Proc. of SIGMOD '12*, pages 97–108, 2012.

M. Yang, B. Ding, S. Chaudhuri, and K. Chakrabarti. Finding patterns in a knowledge base using keywords to compose table answers. *Proc. VLDB Endow.*, 7(14):1809–1820, 2014.

P. Yin, Z. Lu, H. Li, and B. Kao. Neural Enquirer: Learning to Query Tables in Natural Language. In *Proc. of IJCAI '16*, pages 2308–2314, 2016.

X. Yin, W. Tan, and C. Liu. Facto: A fact lookup engine based on web tables. In *Proc. of WWW '11*, pages 507–516, 2011.

C. Zhai and S. Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. Association for Computing Machinery and Morgan &#38; Claypool, New York, NY, USA, 2016. ISBN 978-1-97000-117-4.

M. Zhang and K. Chakrabarti. Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In *Proc. of SIGMOD '13*, pages 145–156, 2013.

S. Zhang and K. Balog. Design patterns for fusion-based object retrieval. In *Proc. of ECIR '17*, pages 684–690, 2017a.

S. Zhang and K. Balog. Design patterns for fusion-based object retrieval. In *Proceedings of the 39th European conference on Advances in Information Retrieval*, ECIR '17, pages 684–690. Springer, 2017b.

S. Zhang and K. Balog. Entitables: Smart assistance for entity-focused tables. In *Proc. of SIGIR '17*, pages 255–264, 2017c.

X. Zhang, Y. Chen, X. Du, and L. Zou. Mapping entity-attribute web tables to web-scale knowledge bases. *Database Systems for Advanced Applications*, pages 108–122, 2013.

Z. Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8:921–957, 2017.

G. Zhou, T. He, J. Zhao, and P. Hu. Learning continuous word embedding with metadata for question retrieval in community question answering. In *Proc. of ACL '15*, pages 250–259, 2015.

S. Zwicklbauer, C. Einsiedler, M. Granitzer, and C. Seifert. Towards disambiguating web tables. In *Proc. of ISWC-PD'13*, pages 205–208, 2013.