Dataset Search and Augmentation

by

Zhiyu Chen

A Dissertation
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy
in
Computer Science

Lehigh University
August 2022

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Zhiyu Chen
Dataset Search and Augmentation

____
Date

____
Prof. Brian D. Davison, Dissertation Director, Chair
(Must Sign with Blue Ink)

____
Accepted Date

Committee Members

____
Prof. Jeff Heflin

____
Prof. Sihong Xie

____
Dr. Dawei Yin

Acknowledgments

It was a wonderful couple of years at Lehigh but all good things must come to an end. The experience during these years is priceless. I would like to take this opportunity to express my sincerest thanks to those who I met on the journey to my Ph.D.

First of all, I would like to thank my advisor, Prof. Brian D. Davison. Without his support, I will not be able to finish my dissertation. Prof. Davison provides me with the freedom to explore all the possible ideas that I was interested in. Through all the failures and successes, I learned how to deal with them calmly, equally and positively. His patience, warm-heartedness and encouragement inspires me to overcome all the difficulties. I will always remember the day he drove me to attend the first job interview in my life when I did not have a driver's license. Without his help, I will not be able to have an internship at Bloomberg. It is my pleasure to be his student.

I would also like to thank my committee members: Prof. Jeff Heflin, Prof. Sihong Xie and Dr. Dawei Yin for their help and advice on my writing of this dissertation and general examination. I would like to especially thank Dr. Dawei Yin who helped my dissertation and provided me with his insightful ideas from a different time zone.

I would like to thank all the people I met during my internships. Michael Liebman from Bloomberg helped me a lot during the onboarding process. He offered me the chance to see how the knowledge in computer science can be used in the financial industry. I would like to thanks Yinan Xu, Ao Luo and Shengfeng Pan from Shenzhen Zhuiyi Technology. I enjoyed those days when we read, shared papers and then implemented new ideas on pre-trained language models quickly. I would like to thanks Jie Zhao, Anjie Fang, Besnik Fetahu, Shervin Malmasi and Oleg Rokhlenko from Amazon. During the internship at Amazon, they provided me with a lot of interesting research ideas and an abundance of computational resources to do experiments.

Many thanks to my labmates from WUME lab: Dan Luo, Youshan Zhang, Sicong

Kuang and Eashan Adhikarla. I enjoyed learning from you during our group meetings. Also thanks to other friends from Lehigh: Mohamed Trabelsi, Yangying Liu, Houliang Zhou, Xiaowen Ying, Hui Ye, Yang Yi, Yujie Ji, Tingzhe Zhou, Xinyang Zhang, Xin Li, Wenbo Li, Qinghan Xue, Dawei Li. I appreciate those days with you at Lehigh. Mohamed Trabelsi is also my collaborator and we worked on multiple papers together. I want to thank Prof. Mooi Choo Chuah who guided me a lot during my first year at Lehigh. I want to thank Shuo Zhang from Bloomberg who spent a lot of time discussing and sharing new ideas with me. I also want to thank Sheng Wang from Wuhan University who discussed with me about dataset search ideas a lot.

I am deeply indebted to my dear father and mother, who have been supporting me from all aspects since I was born. Without them, there will be no me. This thesis is dedicated to them.

Much of the work presented in this dissertation was supported in part by a Lehigh University internal Collaborative Research Opportunity grant and a grant from the National Science Foundation under award IIS-1816325. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Lehigh University or the National Science Foundation.

Dedication

To my parents, Hongwu Chen & Xiaoni Qin.

Contents

List of Tables

List of Figures

Abstract

Data has become an indispensable part of our life. However, current mainstream commercial search engines do not support specialized functions for dataset search. A dataset usually consists of both metadata and data content. Existing information retrieval models designed for Web search cannot efficiently extract semantic information inside structured datasets. Developing new algorithms for next-generation search engines to efficiently find datasets can benefit data practitioners in their data discovery experience.

In this dissertation, we consider how to effectively perform dataset search and augmentation. We start by providing an end-to-end description of a dataset search engine following the lifecycle of datasets. Our review includes web dataset acquisition techniques, dataset profiling and augmentation methods, and dataset search tasks and corresponding methods. In order to extract datasets from research articles, we present an information extraction framework to determine triples of interest which can be used for academic dataset search. We also propose a feature-based method to augment tabular datasets with additional schema labels to help users and systems to better understand the datasets. We develop three methods for tabular dataset search: the first one utilizes generated schema labels to enhance the search results; the second one adopts the pretrained language models to learn the matching features; the third one models the complex relations in the datasets as one or more graphs and uses graph neural networks to learn representations of queries and tables. To support dataset search in which a query is also a dataset, we propose universal dataset encoders which regard a dataset as a point set so that the encoded dataset representations can be used to search for similar datasets. Extensive experiments across multiple tasks demonstrate the superiority of our proposed methods over the state of the art.

Chapter 1

Introduction

1.1 Overview

In the era of big data, many people rely on datasets for their work: data journalists need datasets to tell a good story and researchers use datasets for their research experiments. Today there are more than 1,000 data repositories [161] across various research disciplines. According to Google's statistics [23], the number of datasets grew from 6M in 2018 to 28M in 2020. With increasing volume of datasets available on the Web, finding desired datasets becomes a non-trivial task for modern search engines.

One of the challenges in dataset management and search is the heterogeneity of datasets. A text document can be a dataset. A set of extracted relational RDF triples from a text document can also be a dataset. A table embedded in a document or a Web page can be a dataset. We use the following definitions to define a dataset and its components:

Definition 1.1.1 (Dataset). A dataset is an entity that consists of dataset content and the metadata describing its dataset content.

Definition 1.1.2 (Dataset Content). The dataset content of a dataset is a collection of related observations, organized in certain format.

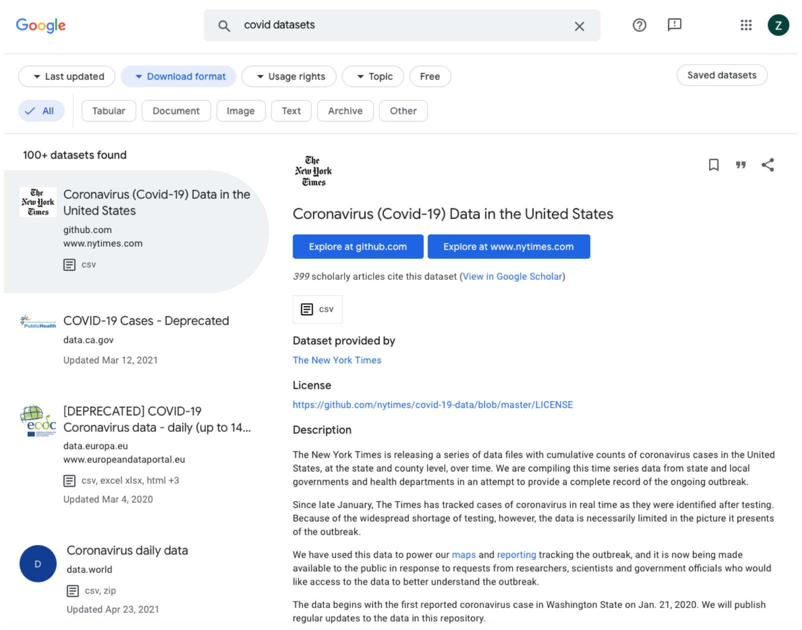
Definition 1.1.3 (Metadata). The metadata of a dataset is the data that provide information about the content of the dataset.

By these definitions, a dataset can be accurately described by its metadata no matter in what format the dataset is stored. Common metadata fields include the name of dataset provider, title, textual description and related hashtags of a dataset. In previous work like Chapman et al. [47], they refer to dataset content defined here as the dataset. However, a collection of related observations or records alone are not enough to describe a dataset. Considering the case where we need to annotate one hundred Yelp reviews for a text classification task and there are two groups annotating the same collection of reviews, it is possible that the two groups agree on all annotated labels. In this case, we obtain two datasets whose dataset contents are the same. But we can still consider them as two distinct datasets since they can have different metadata describing who are the annotators. Metadata is essential to resolve dataset duplication and provenance [34].

Different data portals use different dataset management systems. For example, the Open Data portal[1] hosted by U.S. General Services Administration uses catalog software called CKAN[2] to manage public datasets. With CKAN, dataset publishers are able to upload datasets in their raw formats, with additional metadata such as title, description and hashtags. However, current data management systems have some limitations. First, only the metadata of datasets will be used to produce ranking results. The reason is metadata consists of multiple text fields, while datasets are stored in raw formats. This metadata-oriented ranking strategy can cause problems when the metadata of a dataset is incomplete or of poor quality. In other words, those frameworks do not support the indexing of dataset content and therefore could lead to inefficient matching between query terms and dataset content (in text format). According to Benjelloun et al. [23], about 37% of datasets are in table format which are stored in CSV files or XLS files. Tables represent relational data compactly and contain rich information. For example, the table headers usually represent high-level concepts, which may not be described in metadata. Without

[1]`https://www.data.gov/`

[2]`https://ckan.org/about/`



100+ datasets found

Coronavirus (Covid-19) Data in the United States
github.com
www.nytimes.com
csv

COVID-19 Cases - Deprecated
data.ca.gov
Updated Mar 12, 2021

[DEPRECATED] COVID-19 Coronavirus data - daily (up to 14...
data.europa.eu
www.europeandataportal.eu
csv, excel xlsx, html +3
Updated Mar 4, 2020

Coronavirus daily data
data.world
csv, zip
Updated Apr 23, 2021

Coronavirus (Covid-19) Data in the United States

Explore at github.com
Explore at www.nytimes.com

399 scholarly articles cite this dataset (View in Google Scholar)

csv

Dataset provided by
The New York Times

License
https://github.com/nytimes/covid-19-data/blob/master/LICENSE

Description

The New York Times is releasing a series of data files with cumulative counts of coronavirus cases in the United States, at the state and county level, over time. We are compiling this time series data from state and local governments and health departments in an attempt to provide a complete record of the ongoing outbreak.

Since late January, The Times has tracked cases of coronavirus in real time as they were identified after testing. Because of the widespread shortage of testing, however, the data is necessarily limited in the picture it presents of the outbreak.

We have used this data to power our maps and reporting tracking the outbreak, and it is now being made available to the public in response to requests from researchers, scientists and government officials who would like access to the data to better understand the outbreak.

The data begins with the first reported coronavirus case in Washington State on Jan. 21, 2020. We will publish regular updates to the data in this repository.

Figure 1.1: The result presentation page of Google Dataset Search.

indexing the content of tables, important matching signals could be missed. Second, there is no unified schema for dataset metadata. Schema.org[3] is widely used by search engines while other open standards for dataset metadata also exist such as the W3C Data Catalog Vocabulary (DCAT)[4]. An ideal dataset management system should enable a dataset search engine to rank datasets from heterogeneous resources which can be described with different metadata standards. In 2018, Google launched their dataset search engine [34] in the context of the entire Web. As shown in Figure 1.1, the returned datasets are from multiple sources. However, as mentioned above, the ranking results are based on the matching of metadata and the attached dataset content are not analyzed. Empowering a dataset search engine with the ability to analyze dataset content such as tables could be challenging but useful. In this dissertation, we focus on how to use dataset content (e.g., tables) for dataset search and metadata augmentation.

[3]https://schema.org/docs/about.html
[4]https://www.w3.org/TR/vocab-dcat/

1.2 Contributions

In this dissertation, we mainly focus on developing new algorithms for dataset search and metadata augmentation. We summarize the major contributions of this dissertation as follows:

- We systematically discuss different aspects about a dataset search engine. Specifically, we review techniques of dataset acquisition that can obtain datasets for dataset search engines, various dataset management tasks that can help search engines better understand the content of datasets, and multiple types of dataset search tasks.

- We propose a dataset acquisition method to extract RDF datasets from research articles. We first propose Machine Learning Progress Ontology (MLPO) which defines data of interest.We show how information extraction techniques can be used to extract datasets defined in MLPO which can support various downstream tasks including academic dataset search.

- We introduce a feature-based method for schema label (table header) generation, which can be considered as a data augmentation method for tabular datasets. Through schema label generation, more common schema labels can be provided to allow for broader schema matches in contexts such as dataset search and data linking.

- We propose a two-stage schema label enhanced ranking framework for table search. In the first stage, a schema label generator is trained to generate additional schema labels for each table column. In the second stage, given a user query, tables are ranked together with generated schema labels.

- We propose a table search method based on a pre-trained language model (BERT). Multiple methods are proposed to encode table content considering the table structure and input length limit of BERT. We also develop an approach that incorporates features from prior literature on table retrieval and jointly trains them with BERT.

- We model the complex relations in a table corpus as one or more graphs and then utilize graph neural networks to learn representations of queries and tables. We show that text-based table retrieval methods can be further improved by graph-based predictions which fuse multiple field-level information.

- We represent a dataset as a point set and propose different dataset encoders to learn dataset representations. We show that dataset representations learned by dataset encoders can be used to for dataset retrieval where a query is also a dataset.

1.3 Organization

The rest of the dissertation is organized as follows:

- Chapter 2 provides an overview of a dataset search engine. We first describe current dataset search engines. Then we discuss methods to obtain datasets that can be indexed by a dataset search engine, various tasks for better dataset management and different forms of dataset search tasks.

- Chapter 3 presents an ontology that can guide dataset extraction from research documents. We show how the extracted data can be used for downstream applications such as academic dataset search. Material in this chapter was published as a poster in 2020 [56].

- Chapter 4 presents a novel feature-based method for schema label generation. We propose various features for a column in a table which are further used in a multi-class classification framework to predict the column's schema labels. Material in this chapter was published as a paper at a workshop in 2018 [54].

- Chapter 5 investigates how to use generated schema labels for dataset search. A collaborative-filtering method is proposed to learn schema label features. After generating schema labels, we propose a mixed ranking strategy to incorporate schema labels into the final ranking results. Material in this chapter was published as a paper in 2020 [55].

- Chapter 6 presents how to use pre-trained language model to extract table features. We propose different table linearization strategies to construct input for BERT. We show the powerful features extracted from BERT can benefit the table search task on multiple datasets. A Web table usually has rich context information such as the page title and surrounding paragraphs. Therefore, we propose a new test collection for Web table search which not only provide relevance judgments for query-table pairs but also query-context pairs for each context field which are ignored in previous test collections. Material in this chapter is from two papers published in 2020 [57] and 2021 [59].

- Chapter 7 proposes a graph-based method for ad hoc table retrieval. By explicitly modeling the table corpus as one or more graphs, we directly encode the structural information of the table which is often ignored by previous methods. The material in this chapter was published as a paper in 2021 [58].

- Chapter 8 presents a representation learning framework for datasets by treating a dataset as a point set. We propose different point set encoders to learn dataset representations which can be used for dataset search.

- Chapter 9 summarizes the contributions of this dissertation and discusses the future directions of dataset search.

Chapter 2

Background

In this chapter, we first describe the current dataset search engines (Sec. 2.1). Then we discuss how datasets can be obtained from the Web (Sec. 2.3). Next, we review the tasks of dataset management (Sec. 2.4) and dataset search (Sec. 2.5).

2.1 Current Dataset Search Engines

Here we introduce the existing dataset search engines that are active and free to users. Due to the heterogeneity of dataset modalities, the main difference among dataset search engines are the types of indexed datasets. For different purposes, the indexed datasets can be very different. We summarize the characteristics of different dataset search engines in Table 2.1.

Similar to Web page search engines, Google Dataset Search[1] returns a list of Web pages containing a dataset. Since a Web page can link to datasets of any type, Google Dataset Search can be considered as returning datasets of generic types. This type of dataset search engine usually relies on metadata for ranking instead of dataset content.

Scholarly search engines are designed for researchers who usually have expert-level domain knowledge and a clear intent of what kind of datasets they are looking

[1]`https://datasetsearch.research.google.com/`

Table 2.1: Different types of dataset search engines.

Dataset Search Engine	**Query Type**	**Dataset Type**	**User Upload**
Google Dataset Search [34]	Keywords	Generic	No
CKAN-based Data Portal [87]	Keywords	Generic	Yes
Data.world [72]	Keywords	Generic	Yes
GeoBlacklight-based Data Porta[62]	Keywords	Geospatial	No
UCR STAR [95]	Keywords	Geospatial	Yes
IEEEDataPort[2]	Keywords	Academic	Yes
PapersWithCode[10]	Keywords	Academic	Yes
Kaggle [63]	Keywords	Academic	Yes
Delve [7]	Keywords	Academic	No
DataLab [279]	Keywords	Academic	No
Auctus [41]	Table/Keywords	Table	No
Juneau [305]	Table	Table	No
sameAs.org [10]	URI	URI	No

for. For example, each dataset in IEEEDataPort[2] is linked to a published paper and has a single Digital Object Identifier (DOI). Compared with a generic dataset search engine, the number of datasets indexed by scholarly search engines is usually smaller. However, with an increasing number of published papers and datasets in recent years, scholar dataset search becomes more and more important for researchers to find high quality datasets for experiments. Different from other dataset search engines, citation network analysis is a core component for building a scholar dataset engine [7].

Some dataset search engines are designed for specific dataset types. Many geospatial data portals (e.g., NYU Spatial Data Repository[3]) are build upon GeoBlacklight [62], which is open-source software for building geospatial dataset search engines. Each geospatial dataset is stored in specific format (e.g., Shapefile, KMZ and GeoTIFF). Datasets indexed by Auctus[4] [41] are in tabular format. Auctus also provides functions like data augmentation. Given a query tabular dataset, it can

[2]`https://ieee-dataport.org/`
[3]`https://geo.nyu.edu`
[4]`https://auctus.vida-nyu.org`

find relevant datasets that can be joinable or unionable. Semantic dataset search engine sameAs.org [10] is designed for finding co-references between different datasets. Given the URI of an entity, it returns co-references in other RDF datasets.

Among all those dataset search engines, keywords are the most important query type since it is natural for users to represent information need. Even for semantic dataset search engines, there are also methods supporting keyword search. However, we find many semantic web search engines are no longer active and we do not list them in Table 2.1. For the cases like Auctus and Juneau in which the query is a dataset, sometimes we also call it dataset recommendation.

2.2 An End-to-end Perspective

In order to present an overview of a dataset search engine, we show an end-to-end pipeline in Figure 2.1. We break down the whole pipeline into four different parts by the lifecycle of datasets:

- *Dataset acquisition* (1-2 in Figure 2.1) refers to the identification or extraction of datasets from the web.

- *Dataset management* (3 in Figure 2.1) provides infrastructures to store and maintain datasets.

- *Dataset search* (4-6 in Figure 2.1) is the task of answering a search query by returning a list of datasets. The query can be either a keyword query or a dataset itself.

- *Data-centric applications* (6-8 in Figure 2.1) rely on the accessibility of a large amount of datasets facilitated by services in dataset management and search.

We describe the details of every component in the rest of this chapter.

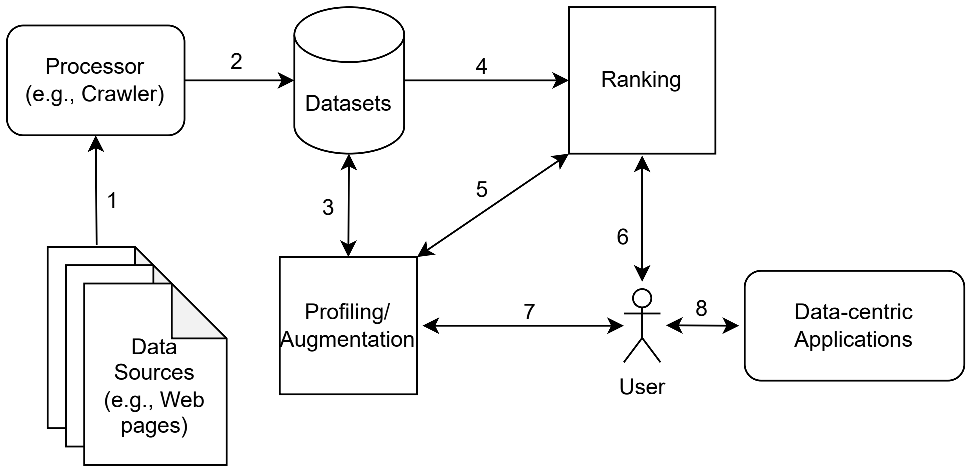

Processor (e.g., Crawler)
2
Datasets
4
Ranking
1
3
5
6
Data Sources (e.g., Web pages)
Profiling/ Augmentation
7
8
User
Data-centric Applications

Figure 2.1: Overall pipeline of a dataset search system.

Table 2.2: A list of dataset dumps available.

Dataset	**Data Type**	**Source**
Metadata of Datasets	metadata	Google Web Crawl [33]
TableArXiv	table	scientific documents [91]
WikiTables	table	Wikipedia [27]
AI-KG	RDF	scientific documents [76]

2.3 Dataset Acquisition

The process of dataset acquisition has a huge impact on the quality of datasets and therefore is an important step for building any downstream data-focused task. For traditional Web page search engines, a Web crawler [176, 120] is designed to download Web pages starting from some seed URLs and then recursively download more Web pages following hyperlinks. Algorithms like PageRank [193] are proposed to determine the importance of Web pages so that the quality of indexed content can be improved. Compared to crawling Web pages, crawling high-quality datasets is much more challenging than crawling Web pages. A URI can be an identifier of a Web page, while the identification of a dataset requires more effort.

Generally, existing dataset search engines have two ways to obtain datasets. The first way is asking dataset providers to upload their datasets associated with metadata in a predefined format. Open dataset portals are designed like this. The advantage is that dataset providers are able to maintain their datasets frequently. The disadvantage is that there is no unified definition for dataset metadata. Schema.org is one of the standards while other open standards for dataset metadata also exist such as the W3C Data Catalog Vocabulary (DCAT)[5]. Certain communities can have their own defined metadata standard. For example, GeoBlacklight uses OpenGeoMetadata schema which is designed specifically for geospatial resource discovery. The second way is to adopt some strategies that allow a dataset search engine to automatically crawl datasets from the Web, just like the Web page search engines. The implementation of Google dataset search [34] is an example. On one hand, automatic dataset acquisition methods (Sec. 2.3.2) can rely on modern machine learning techniques so that the dataset search engines can scale to all potential datasets on the Web. And less human curation is required to maintain the increasing number of datasets. On the other hand, more complex dataset acquisition methods should be designed so that the quality of datasets is reliable.

In general, there are two cases where a dataset is located:

1. **Dataset Links**: a dataset appears as a downloadable link which refers to the real dataset file;

2. **Embedded Datasets**: a dataset is embedded in a document (e.g., in an HTML file or a PDF file).

For dataset links, it may not be feasible for search engines to index and analyze dataset files or dataset content according to our definition 1.1.1, because the target dataset files can be in any format or even inaccessible in the worst case. However, it will still be helpful for a user to find a Web page that potentially contains the link to download a dataset for further review. The information appearing on the same page can be saved as the metadata of that actual dataset file. For embedded datasets such as Web tables, certain information extraction methods [93, 5, 300] can be used to extract datasets and dataset content is directly accessed.

[5]`https://www.w3.org/TR/vocab-dcat/`

Dataset acquisition is also an important topic in data management community [92, 9] and machine learning community [224]. Unlike prior work discussing how to acquire datasets used in the data integration process or how to use dataset crowdsourcing methods for specific machine learning applications, we discuss dataset acquisition from the perspective of a dataset search engine developer. In this section, we discuss various methods that have already been used in existing dataset search engines or can potentially be used in the future to acquire datasets for building an open dataset search engine. Because different types of datasets are from different sources, we categorize those methods by data types. We first talk about how to make use of an existing search engine to crawl Web pages that contain datasets. Then we discuss different methods about how to extract tabular datasets either from the Web or documents. We summarize a list of available dataset dumps that are ready to be indexed in Table 2.2.

2.3.1 Dataset-focused Crawling

Existing crawlers of Web page search engines can be helpful and adapted to crawl datasets. Topic-focused crawlers [43, 78, 164, 186, 19, 260] and Web page classification [97, 116, 3, 229, 210] have been studied to effectively narrow down the crawling boundary and process links focused on specific topics. WEBTABLES [36] and OCTOPUS [35] simply rely on an existing search engine to obtain the candidate Web pages and then those without tabular datasets are filtered. TableSeer [151] crawls the web pages using a depth-first crawling policy with a maximum depth of five from the seed URLs. A classifier is used to determine whether a document contains tables or not. Documents with tables are further processed so that tables and metadata such as page title, table caption and table headers are indexed.

More recently, Zhang et al. [295] propose a domain-specific dataset discovery system (DSDD) where given a user keyword query, a list of potential Web pages containing at least one dataset link are returned. DSDD first discovers dataset repository entry pages (DREP). Based on a search engine, it mimics a user's dataset discovery process and schedules a list of predefined actions (including keyword search and related pages search using a search engine, forward and backward crawling) until a certain number of pages are analyzed. An online algorithm based on multi-armed bandits [12] is trained to select the best action that maximizes the harvest rate and coverage of DREP. For each crawled Web page, a SVM classifier trained with TF-IDF features is used to classify whether the Web page is a DREP or not. To further increase the accuracy of crawling results, a verification step is performed if a page is classified as a DREP. Starting from the candidate entry page, it crawls all subsequent pages within three hops. For each page, regular expressions are used to check for URLs that link to dataset files. After a DREP page is verified, DSDD further crawls dataset pages (DP) that points to dataset files (e.g., CSV, TSV, JSON, ZIP).

The identification of datasets can be easier when Web developers provide metadata information in the source code following certain specifications[6]. Google Dataset Search [34] relies on Google Web crawling which processes structured data markup and generates triples for each page. Specifically, all the pages containing the following types of elements can be identified as pages containing datasets: `http://schema.org/Dataset`, `http://schema.org/DataCatalog`, and `http://www.w3.org/ns/dcat#Dataset`. Based on the standard and structured data markup, the metadata of datasets are directly crawled. Those raw metadata are further cleaned and indexed by Google's Dataset Search Engine. An obvious defect of this approach is that the coverage of embedded metadata matters since those Websites without those markup cannot be discovered. According to Benjelloun [23], about 500k Web pages included dataset markup and half of them were from the US Open Government portal. But until March 2020, there are about 28 million datasets from more than 3,700 sites. The increasing number indicates the huge impact of using semantic markup in the future.

[6]`https://developers.google.com/search/docs/advanced/structured-data/dataset`

2.3.2 Acquisition of Tabular Datasets

Tables are datasets organized in a structured format. Many modern software applications provide versatile functions to work with tabular datasets (e.g., Microsoft Excel). Tabular datasets can be easily transformed into RDF datasets which are widely used in the semantic web. Ding et al. [79] show that tabular datasets from Data.gov can be converted into RDF triples and then linked to different data sources like the Linked Open Data Cloud (e.g., DBpedia), conventional web (e.g., New York Times), and their own provenance annotations. Methods of table extraction from the Web are important approaches to dataset acquisition, since tabular datasets are the most used type of datasets in the world and can be easily converted into other formats.

Web Table Extraction

The Web contains a large number of datasets which are embedded as HTML source codes. On the top right-hand corner of many Wikipedia pages, there is a infobox which presents the summarized facts of that page formatted in a table. Such tabular datasets are very useful for downstream applications such as knowledge base construction [197, 162, 31, 2] and question answering [115, 1, 171]. Web tables in general can have complex layout structures for different purposes [269, 37, 65, 134, 53, 139]. The <table> tag is frequently used to display information facilitating better visual effect. It is also used to organize a list of facts such as Wikipedia infoboxes. Tables which include rich information about entities and their attributes are also called relational tables. Extracted relational tables can be transformed/normalized into a consistent format. The core-column [138, 299, 27, 259] which refers to a special column that includes important entities is usually detected by algorithms [32]. A normalized table is a set of cells arranged in rows and columns like a matrix. Each cell could contain a single word, a real number, a phrase or even sentences. The first row of a table is the header row (can be empty) and consists of header cells. The table itself is the dataset content of tabular data. The metadata of a table usually includes the caption of the table, the title of the Web page containing the table.

Sometimes the the table headers are also considered as metadata. A relational table can also be transformed into semantic triples in the form of $< p, s, o >$, where p is a predicate, s is the subject and o is its object [65, 79].

Early work [48, 288] proposes different rule-based algorithms to detect Web tables for a small set of data or domain-specific tables. Wang et al. [269, 270] explore various features reflecting the layout and content characteristics of tables. Simple machine learning methods such as naive Bayes model and KNN show good performance on Web tables from multiple domains. As the first stage of the *WEBTABLES* project, Cafarella et al. [37] extracted 14.1 billion Web tables from Google's Web crawl and about 154 million of them are high-quality relational tables. They first wrote a list of filters to remove non-relational tables and then employed a similar method to Wang et al. [270] to further classify non-relational tables and table headers. Crestan and Pantel [65] propose a fine-grained taxonomy instead of a binary classification of table types. They further propose several groups of features for table type classification: the global layout features account for the structure of the table as a whole; layout features and content features are generated over rows and columns. A Gradient Boosted Decision Tree (GBDT) [88] is trained to classify a Web table into one class in the fine-grained taxonomy. For different types of tables, the position of core-columns are different. Lautert et al. [134] extends the taxonomy defined in Crestan et al. [65]. Five additional features are proposed which are position of inner HTML tables and ratio of cells containing unordered lists, ordered lists, commas and brackets. They use a Multilayer Perceptron Network with 1 hidden layer as the table type classifier.

Data Extraction from Documents

Documents, especially scientific articles, are also important source for extracting datasets. Current scholarly search engines such as Google Scholar[7], CiteSeerX[8] and Semantic Scholar[9] support full-text ranking to find relevant documents given a user

[7]`https://scholar.google.com/`
[8]`http://citeseerx.ist.psu.edu`
[9]`https://www.semanticscholar.org/`

query. With an increasing number of publications each year, full-text search can be less useful when people have a more specific information need. For example, an NLP researcher may want to find the most recent publications on a specific dataset/task and rank them by their performance instead of relevance to a simple keyword query. PapersWithCode[10] provides similar services to facilitate the growth of the machine learning community. In the following, we discuss two types of data that can be extracted from documents to support future dataset search engines.

Tables in scientific articles are explicit datasets like Web tables which summarize important research results in a compact way. Table detection and recognition in PDF files or images is also a well-studied research direction [75, 212, 282]. There are usually two steps towards obtaining tabular datasets from docoments: (1) identifying the region in a document that encloses the table; (2) extracting the structural information like rows and columns from the table. Many open-source tools are available for table extraction from documents such as Camelot[11], Tabula[12] and PDFPlumber[13]. TableSeer [152], a search engine for tables, proposes its own algorithm of extracting tables associated with metadata from scientific articles. Gao et al. [91] construct TableArXiv[14], which is a collection of scientific tables. They used a snapshot of arXiv[15] containing 854,989 papers. A LaTex processing tool LaTeXML [238] is used to convert papers' LaTeXsource files to an XML format, which are further used to extract tables. In total, they extracted 341,573 tables from papers in Physics-related domains.

Implicit datasets can also be extracted from documents. With the help of information extraction techniques, structured data can be extracted from unstructured text. For example, the task of extracting useful tuples from publications have gained more and more attention in recent years. Such data can be helpful for tracking the progress of different research communities. Luan et al. [155] propose a multitask

[10]https://paperswithcode.com
[11]https://github.com/camelot-dev/camelot
[12]https://github.com/chezou/tabula-py
[13]https://github.com/jsvine/pdfplumber
[14]http://boston.lti.cs.cmu.edu/eager/table-arxiv/
[15]http://arXiv.org

framework *SCIIE* to identify and classify scientific entities, relations, and coreference resolution across sentences. Hou et al. [104, 105] propose a framework *TDMS-IE* aiming at extracting <Task, Dataset, Metric Name, Metric Score> tuples from NLP papers. The tuples can be used to automatically build NLP leaderboards. With similar purpose, Kardas et al. [118] propose AxCell to extract results of papers. The results of AxCell is used by the leaderboards on the PapersWithCode platform. Wu et al. [278] propose a new way of information extraction which adopts a seq2seq approach to transform text into tables. Practically, the extracted tuple targets can be different depending on specific applications. Different ontologies [76, 56] have been proposed which define various entity types and relation types. The resulting tuples can be transformed into RDF triples and queried via a SPARQL endpoint.

2.4 Dataset Management

After obtaining the datasets, a management system is required to index and store them. More advanced tasks such dataset profiling are important to enable effective downstream tasks such as dataset search and recommendation. In this section, we specifically discuss about dataset profiling and dataset quality control. We show a list of open source tools for dataset management in Table 2.3.

2.4.1 Dataset Profiling

Dataset profiling has been studied by the Semantic Web community [4, 20] for decades. However, we do not limit the scope of discussion to only RDF datasets. Prior works on RDF dataset profiling are instructive for profiling general datasets, especially for datasets that can be converted into RDF triples. Therefore, the methods discussed here can be generalized to datasets of any format that have a tabular correspondent. Here we give the definition of dataset profiling:

Definition 2.4.1 (Dataset Profiling). Dataset profiling is a process of analyzing the raw datasets and generating additional information about the original datasets.

Table 2.3: A list of opensource tools for dataset management.

Management Tool	Data Type	URL
OpenClean [173]	Table	https://github.com/VIDA-NYU/openclean
Auctus [41]	Table	https://github.com/VIDA-NYU/auctus
DDT [130]	Table	https://github.com/VIDA-NYU/domain_discovery_tool
DataHub	Metadata	https://github.com/linkedin/datahub
DATALAB [279]	Scholar	https://github.com/ExpressAI/DataLab
Pytheas [60]	Table	https://github.com/cchristodoulaki/Pytheas
Valentine [127]	Table	https://github.com/delftdata/valentine
DataTypes [255]	Table	https://github.com/ivaleraM/DataTypes
Duke [15]	Table	https://github.com/NewKnowledge/duke
Sherlock [109]	Table	https://github.com/mitmedialab/sherlock-project
D4 [190]	Table	https://github.com/VIDA-NYU/domain_discovery_tool
Sato [294]	Table	https://github.com/megagonlabs/sato
Juneau [305]	Table	https://github.com/juneau-project/juneau
Doduo [239]	Table	https://github.com/megagonlabs/doduo

Basic dataset profiling features that can be directly calculated from datasets include the maximum/minimum values and the number of values or distinct values of a column [41]. Castelo et al. [41] use K-Means clustering algorithm to calculate dataset summaries where the ranges of their corresponding attributes are represented. Such profiling results are also used in result presentation of some dataset search engines. In the following, we introduce more advanced profiling tasks that require explicit modeling with machine learning methods.

2.4.2 Attribute Annotation

The headers of tabular datasets sometimes are also called dataset attributes. However, many datasets have incomplete information and headers can be missed. There are two types of profiling tasks that can provide additional information about dataset attributes. The first one is **statistical type annotation** which discovers the statistical data types (e.g., ordinal, categorical or real-valued). Valera et al. [255] propose a Bayesian method to solve this problem by exploiting the latent structure in the data. They distinguish among real-valued, positive real-valued and interval data as types of continuous variables, and among categorical, ordinal and count data as types of discrete variables. Ptype [42] uses Probabilistic finite-state machines (PFSM) that can generate values from the corresponding domains. Then given a column of data values, the column type can be inferred. Bonfitto et al. [29] infer the types of columns in CSV tables by exploiting a decision tree to do multi-label classification.

Beyond annotating columns with simple data types like integer, string, date, and float, **semantic type annotation** aims at annotating columns with meaningful concepts either from a knowledge base (e.g., DBpedia classes) or datasets themselves. *WEBTABLES* [36] can finds potential synonyms of a given table header from corpus-wide statistics on co-occurrences of table attributes. Limaye et al. [147] model the table annotation problem using a number of interrelated random variables and propose a probabilistic graphical model to annotate one or more types for a table column. Barcelos [64] builds a regression model to rank candidate annotations. Given a target column, columns with overlapping values are retrieved as candidates. Two group of features are used to fit the regression model. The first group calculates Jaccard similarity between values of two columns. The second group calculates the Jaccard similarity of other attribute names. ColNet [49] adopts Convolutional Neural Networks (CNNs) to embed the overall semantics of a column into a single vector which captures both inter-cell and intra-cell locality features. Ota et al. [190] derive robust context signatures of columns from a large number of datasets. A column-based clustering approach is used to find attribute candidates belonging to the same domain. Manually curated features capturing various statistics of columns are used to fit a random forest for attribute prediction in [54, 287]. Sherlock [109] additionally obtains two types of embedding features for columns. The first one is obtained by calculating the mean, mode, median and variance of GloVe embeddings [198] across all values in a column. The second one is considering a column as a paragraph and training paragraph vectors [135] for columns. Then both statistical features and embeddings features, which capture character-level, word-level and column-level signals, are used to train a neural network for semantic type annotation. Zhang et al. [294] uses Sherlock [109] as the single-column prediction model and explores structured learning to perform multi-column prediction, where table-wise context and column-wise context are used. More recently, pre-trained models have been used for semantic type annotation. Deng et al. [74] propose a Transformer-based framework *TURL* to model the row-column structure of relational tables. Similar to masked language modeling used in BERT [77], they propose a pre-training task to recover the masked entity cells. *TURL* can be used as a feature extractor of columns for the semantic type prediction task. Trabelsi et al. [248] adopt BERT to generate headers sequentially for a table so that the context information is considered. Suhara et al. [239] serialize a table into a sequence and use BERT to extract column-wise features and column-pair features. Then a multi-task framework is utilized to solve semantic type annotation and column relation prediction simultaneously. In Chapter 4, we introduce our feature-based method for column attribution annotation in details.

2.4.3 Entity linking

Datasets may have potential entity mentions (e.g., persons, locations, etc.) that can be linked to knowledge bases (KBs). Uncovering or extending such semantics of datasets could be helpful for tasks like dataset retrieval, table-based question answering and dataset augmentation. Usually there are two steps in entity linking from a dataset: (1) a list of candidate entities are retrieved; and, (2) an entity disambiguation module is used to rank and select the matched entity. For candidates retrieval, an existing Wikidata lookup service can be used as in [27, 219, 83].

The probabilistic graphical model proposed by Limaye et al. [147] can not only annotate table attributes, but also annotate table cells with entities defined in knowledge bases. *TabEL* [27] assumes that entities in a given row or column tend to be related and utilize a collective classification technique to encourage disambiguation of mentions in the same row or column to be related to one another. The disambiguations in a given table are optimized jointly in order to obtain a globally coherent set of entities. Wu et al. [277] propose to link entities with multiple linked KBs. First, they use a graph-based algorithm to link entities from each single KB.

Then several heuristics are presented to leverage "sameAs" relations between entities from different KBs to improve the entity linking results from previous step. Ibrahim et al. [111] give specific consideration to quantities and distinguish between numerical mentions and string mentions. They build a new knowledge base *QKB* for quantities. A quantity is a triple <*measure*, *value*, *unit*> where *measure* refers to a certain quantifiable aspect of an object or process (e.g., height, revenue), *value* is a numerical literal and *unit* represents the magnitude of a quantity. *TURL* [74] treats each cell of a table as a potential entity. The representation of a candidate entity retrieved from Wikidata Lookup service is calculated from *TURL*'s Transformer encoder using its name, description and type. A similarity function is fine-tuned with cross-entropy loss to predict the matching score between the cell and candidate entity.

2.4.4 Vectorization

Vector representations for documents have been studied for text understanding. From early Bag of Words (BOW) representations [228] to recent dense representations learned from neural networks [50, 119, 293, 280, 292], vectorization of documents is important for retrieval tasks. The indexed vectors are efficient for search engines to compute their similarity or relevance to a query. Here, we also consider dataset vectorization as a profiling task according to Definition 2.4.1. A dataset management system can learn vector representations for datasets offline which can support downstream tasks such as dataset retrieval.

The same vectorization techniques for text documents can also be used for datasets. DataLab [279] encodes the descriptions of datasets with BERT and the embeddings of [CLS] tokens are treated as vector representations of datasets. Some vectorization methods are proposed specifically for tabular datasets. Inspired by the Word2Vec approach [167], Table2vec [296] trains four types of table embeddings based on the co-occurrences of different table elements. The first considers all words appearing in a table and its metadata fields (e.g., page title, caption, etc.). The second only considers table headers and each header is treated as a single term. The third considers entities appearing in the core-column and the last one considers all entities from table cells. Cappuzzo et al. [38] formulate dataset embeddings learning as a graph embeddings generation problem. First, they transform a dataset into a compact tripartite graph with three types of nodes. *Token* nodes represent each token appeared in the dataset. *Record Id* nodes (RIDS) represent a tuple in the dataset. *Column Id* nodes (CIDs) represent attributes of the dataset. Edges are constructed based on the structural relationships in the datasets. Random walks are used to generate sentences from the constructed heterogeneous graph. Then embedding training algorithms such as Word2Vec can be used to train vector representations for each node in the graph. For methods proposed in [296, 38], a dataset is vectorized into a list of embeddings of dataset elements. Those embeddings can be indexed as dataset features to train more complex representation learning models in downstream tasks. Or simple strategies such as averaging can be used to derive a single representation for each dataset.

2.4.5 Quality Control

The quality of datasets varies due to many reasons. A five-star scheme for grading the quality of linked data was proposed by Tim Berners-Lee [25]:

- **One Star**: available on the Web (with an open licence) to be Open Data;
- **Two Star**: available as machine-readable structured data (e.g., excel instead of image scan of a table);
- **Three Star**: as **Two Star** plus non-proprietary format (e.g., CSV instead of excel);
- **Four Star**: all the above plus, use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff;
- **Five Star**: all the above plus, link the dataset to other people's datasets to provide context.

The standards can be adapted a little bit to measure quality of datasets in other formats. For a **Four Star** dataset, instead of requiring it to use open standards to identify things, we can expect the dataset is described with open standard metadata definitions (e.g., `Schema.org`) so that those datasets can be easily discovered and accurately interpreted by dataset search engines. However, there are different metadata definitions and there is still a long way to go before all the datasets can be described with a unified metadata definition.

Metadata is documentation of datasets. The machine learning community has proposed different standards to document datasets. Gebru et al. [94] propose datasheets for datasets in order to facilitate better communication between dataset providers and consumers. They provide a template which contains a list of questions about the dataset creation, maintenance, and distribution process. Pushkarna et al. [203] propose Data Cards for datasets which summarize essential facts about datasets such as upstream sources, data collection and annotation methods, training and evaluation methods, intended use, factors that can affect a model's performance and so on. Domain specific metadata standards have also been proposed. For example, Healthsheet [225] is adapted from Gebru et al. [94] to document healthcare datasets (e.g., electronic health records (EHR), clinical trial study data, etc.).

There are also post-processing techniques for dataset quality control. For example, traditional data cleaning tools can be used [180, 211, 117, 246, 217, 8, 173] to detect errors or repair data in dataset content. Pytheas [60] provides functions to normalize CSV tables. Google dataset search [34] performs a number of operations to clean metadata of datasets. First, they notice that the usage of `Schema.org` properties has different patterns. So for important properties, they write adapters to normalize them into the same pattern. They also construct a knowledge graph from dataset metadata where important entities such as organizations are connected. The crawled datasets from the same or different websites can include many duplicates or replicas. One solution to identify duplicates and replicas is through `http://schema.org/sameAs` if specified explicitly by Web developers. Some heuristics can also be helpful. For example, if two datasets shares a large part of their metadata or point to the same download URL. A graph-based solution is also proposed in Brickley et al. [34] to identify duplicates and replicas at scale. For each dataset, they compute a hash value (fingerprint) for dataset title, description and URL, respectively. Then a graph can be constructed where nodes represent datasets and an edge connects two datasets if they share at least one fingerprint. A MapReduce version of a connected-component algorithm [213] is used to identify connected components. Each connected component is a cluster of duplicates if the datasets are from the same site, or replicas, if they are from different sites.

2.5 Dataset Search

Dataset search is inherently an information seeking procedure which has been studied by different research communities for a long time. For example, building retrieval models to search over RDF datasets [85, 102, 11] is an important task in the semantic web community and document search [222, 251, 252] has been extensively studied by the information retrieval community. We have the following definition for dataset search:

Definition 2.5.1 (Dataset Search). Given a query q, the goal of dataset search is to rank a set of datasets $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ in descending order of their relevance scores with respect to q.

Prior work in dataset search mainly answers a user query by retrieving only one type of dataset. For example, the majority of current dataset search engines listed in Table 2.1 return datasets in specific format. Though the Google Dataset Search returns datasets of any type, only the text description in metadata is analyzed. It is an extremely difficult task to build an universal dataset search engine that can handle all different modalities. Therefore, we discuss different types of dataset search by the type of query or the dataset's modality.

2.5.1 Search by Metadata

Metadata Matching. Current dataset search engines mainly rely on metadata to match user queries. For example, the Google Dataset Search Engine [34] indexes only metadata extracted from Web pages that employ dataset markup defined by Schema.org. Given a user query, the same methods used in the Google Web search engine are applied to search datasets. Since each metadata comes from a certain Web page, the task of dataset search is converted to traditional Web page search. Similarly, all dataset search engines implemented based on CKAN [87] use Apache Solr[16] to support the dataset search functions. A user query is matched against one or more metadata fields with text matching methods.

DataLab [279] is a data-oriented platform mainly designed for NLP researchers. The datasets are also searched using metadata. Instead of encoding dataset content, they use BERT [77] to encode the descriptions of datasets extracted from papers to obtain dataset representations. When the system receives a user query, BERT is also used to obtain the query representation so that dataset representations and query representations are in the same semantic space. Then datasets can be ranked by approximate nearest neighbor search in the semantic space.

Dataset Navigation. In some literature such as Nargesian et al. [177] and Ouellette et al. [191], there is no explicit keyword query from a user. Instead, they assume a user directly navigates the metadata of datasets organized in certain structure to help users find a targeted dataset. Nargesian et al. [177] extract attributes including keywords, concepts and entities from metadata. Then they construct an organization for datasets where a node represents an attribute of a datasets and each dataset can be associated with one or more attributes. Their goal is to construct an organization for datasets so that the expected probability of discovering datasets are maximized. Through a user study, they find that the navigation approach can help users find datasets that are more diverse than keyword search and 42% preferred the use of navigation over keyword search. An interesting observation is that there is only about 5% intersection between datasets found using navigation approach and keyword search, which indicates that the two types of dataset search method can be complementary to each other.

[16]`https://solr.apache.org/`

Discussion. Searching datasets through metadata is the first trial of dataset search, because many existing Web search techniques can be directly applied. However, there are also some limitations. First, a metadata-oriented ranking strategy can cause problems when the metadata of a dataset is incomplete or of poor quality. Second, those frameworks do not support the indexing of dataset content, which could lead to inefficient matching between query terms and dataset content (in text format). Third, there is no unified schema for dataset metadata. Schema.org is widely used by search engines while other open standards for dataset metadata also exist such as the W3C Data Catalog Vocabulary (DCAT)[17]. Therefore, an ideal Web-scale dataset search engine should be able to rank datasets from heterogeneous resources. Empowering a dataset search engine with the ability to analyze dataset content could be challenging but useful.In the remaining subsections, we discuss the search techniques that utilize different dataset content.

2.5.2 Tabular Dataset Search

Tabular dataset search aims to find relevant datasets whose data contents are in table format. According to Benjelloun et al. [23], datasets in tabular format (e.g., in CSV or XLS format) are the most common type of data (37%) and structured datasets (e.g., in JSON or XML format) are the second most common (30%). Any other type of datasets such as images and text are no more than 5%. As we mentioned in Section 2.3, datasets in tabular format and other structured format can be converted into each other, which represent more than half of discoverable datasets without considering there are a large number of datasets that can be extracted from the Web.

Unsupervised Methods. Under the setting of unsupervised learning, annotated relevance scores of query-table pairs are not used for model training. TableRank [151] uses a modified version of a vector space model *TF-IDF* [228] to represent a user query and a table. The relevance of a table given a query is scored by the

[17]`https://www.w3.org/TR/vocab-dcat/`

cosine similarity function. Cafarella et al. [36] propose the WEBTABLES system which allows users to search tables with a keyword query as the input. The table ranking strategy is implemented on top of an existing search engine and returns the top-k extracted tables from the results. OCTOPUS [35] is an extended version of WEBTABLES. It first utilizes a commercial search engine to obtain a list of candidate tables and then reranks the tables with SCPRank algorithm. SCPRank measures the correlation between a query q and a table cell c with the following equations:

$$scp(q,c) = \frac{p(q,c)^2}{p(q)p(c)} \tag{2.1}$$

$$p(q,c) = \frac{\text{number of web pages contain both } q \text{ and } c}{\text{total number of web pages}} \tag{2.2}$$

$$p(c) = \frac{\text{number of web pages contain } c}{\text{total number of web pages}} \tag{2.3}$$

where $p(q)$ is defined in a similar way. Given a user query q and a table of n columns, Equation 2.1 is used to score each column independently by summing the scp scores of all cells of a column. Then the final ranking score of a table is the maximum of all of its per-column scores.

Both WEBTABLES and OCTOPUS rely on an existing search engine to obtain the table ranking results, which means a table along with the context information is treated as an ordinary document. Therefore, any information retrieval method can be applied to tabular dataset search. Zhang et al. [299] propose single-field document ranking where a flattened table and its context fields are concatenated. Then traditional methods such as BM25 or language models can be used to score a query-table pair. Instead of collapsing all fields into a single field, they also propose multi-field document ranking where each field is scored independently and then scores of different fields are aggregated into the final ranking score:

$$score(q,T) = \sum_i w_i \times score(q,f_i) \tag{2.4}$$

where w_i is the weight of field i, f_i is the text representation of field i, *score* is the scoring function such as BM25 or language models. Gao et al. [91] propose a probabilistic framework to retrieve scientific tables extracted from research publications. They first build a structured query from an unstructured query by extracting target quantity types and key concepts from an external knowledge base such as Wikipedia. A table is represented as a structured object that has multiple fields or representations. In the probabilistic framework, they assume different parts of the query are conditionally-independent representations which are matched with differing weights to different parts of the table to make use of all of the evidence available.

Mismatch between a user query and a tabular dataset is one of the challenges in dataset search. For example, a user could search for the locations of Point-of-Interest in New York City with a query "NYC POI locations". A target dataset could include a table with a header named "Locs" which is the abbreviation of "locations". Such abbreviations appear frequently in datasets. In order to relieve this situation, we propose a method in Chapter 4 to generate alternative table headers so that the table headers, which usually are important concepts, have a higher chance to be matched with user queries. Manually curated features from a column are extracted and the alternative header generation is formulated as a multi-class classification problem. We further propose to rank a dataset by considering its metadata, dataset content and augmented metadata in Chapter 5. More than the features used in Chapter 4, we also use CoFactor method [146] to learn latent features from the co-occurrence relationships of table headers. After generating alternative table headers, we use BM25 to score original table headers, table content and generated table headers, respectively. Then those scores are summed up as the final ranking score for a tabular dataset.

Supervised Methods. In contrast to unsupervised setting, supervised table search methods require annotated relevance scores of query-table pairs during the training phase. Cafarella et al. [36] propose featureRank which uses the table-specific features (features 1-8 in Table 2.4) to score each table and does not rely on an

Table 2.4: Features proposed by Cafarella et al. [36].

ID	Description	Dim.
1	number of rows	1
2	number of columns	1
3	has header or not	1
4	document-search rank of source page	1
5	number of hits on header	1
6	number of hits on second-to-leftmost column	1
7	number of hits on leftmost column	1
8	number of hits on table body	1
9	schema coherency score	1

existing search engine. A linear regression model is trained to fit the proposed features. They further propose schemaRank which includes schema coherency score (feature 9 in Table 2.4) as an additional feature. The coherency score measures how well attributes of a schema (table headers) fit together. Given the table header $H = \{h_1, ..., h_n\}$ of table T, its schema coherency score is calculated as:

$$cohere(T) = \frac{\sum_{h_1 \in H, h_2 \in H, h1 \neq h2} PMI(h_1, h_2)}{|R| * (|R| - 1)} \tag{2.5}$$

$$PMI(h_1, h_2) = log(\frac{p(h_1, h_2)}{p(h_1)p(h_2)}) \tag{2.6}$$

where the definitions of $p(h_1, h_2)$,$p(h_1)$ and $p(h_2)$ are similar to Equation 2.2 and 2.3. Pointwise Mutual Information (PMI) is often used in computational linguistics and measures how strongly two items are related. The score will be large and positive when two variables strongly indicate each other, zero when two variables are completely independent, and negative when variables are negatively-correlated. The coherency score is the average of all possible header-pairwise PMI scores and rewards tables with highly-correlated attributes. Similar to featureRank, schemaRank fits a linear regression model with the proposed features (1-9 in Table 2.4).

Zhang et al. [299] propose a Learning-To-Rank (LTR) approach which extends the features in Table 2.4 with more table structure and lexical features listed in

Table 2.5: Features proposed by Zhang et al. [299]

ID	Description	Dim.
10	Number of query terms	1
11	Sum of query IDF scores in field f	number of fields
12	The number of empty table cells	1
13	Number of in-links to the page embedding the table	1
14	Number of out-links from the page embedding the table	1
15	Number of page views	1
16	Inverse of number of tables on the page	1
17	Ratio of table size to page size	1
18	Ratio of the number of query tokens found in page title to total number of tokens	1
19	Ratio of the number of query tokens found in table title to total number of tokens	1
20	Rank of the table's Wikipedia page in Web search engine results for the query	1
21	Language modeling score between query and multi-field document repr. of the table	1

Table 2.5. Note that feature 21 is obtained from Equation 2.4. A random forest is used to fit the ranking features in a pointwise manner. In order to go beyond lexical matching, they further propose a Semantic-Table-Retrieval (STR) approach which represents both queries and tables in a semantic space, and then measure the similarity of those vector representations. First, a table (or query) is mapped into 4 different vector representations:

- **Bag-of-Entities**: To obtain the bag-of-entities representation for a query, the query is issued against the DBpedia knowledge base to retrieve top 10 entities. While for a table, its caption and page title are used as the query to retrieve entities, respectively. If a table has a core column, the linked entities in that column are also added into the bag-of-entities representation of a table.

- **Bag-of-Categories**: Since each entity in a bag-of-entities representation has a corresponding Wikipedia category, the bag-of-categories representation for a query (or table) can also be obtained.

- **Word Embeddings**: Each word in a query (or table) is mapped to a pre-trained word embedding (e.g., word2vec with 300 dimensions, trained on Google News.)

- **Graph Embeddings**: Each entity in a query (or table) bag-of-entities representation is mapped to a pretrained graph embedding (e.g., RDF2vec [218] with 200 dimensions, trained on DBpedia 2015-10).

For each type of the four semantic spaces, there are four similarity measures (listed in Table 2.6) to calculate the similarity score for a query-table pair, which result in $4 \times 4 = 16$ semantic features. Along with all other features proposed in Table 2.4 and Table 2.5, a random forest is trained in a pointwise manner. Tables are also represented as graphs by Trabelsi et al. [250] in which a knowledge graph representation indicates the relations between entities. R-GCN [230] is applied on knowledge graphs to learn representations for graph nodes and relations.

In Chapter 6, we propose a method based on the pretrained language model BERT [77] for tabular dataset search. We first slice a table into rows, columns or cells. A content selector is designed to choose the most significant pieces as BERT input. BERT can be used as a feature extractor and trained jointly with previously designed features [299] to achieve significant improvement. A graph-based approach is proposed in Chapter 7. We first construct word co-occurrence graphs for metadata and data content separately. Then graph neural networks are used to learn the embeddings of nodes, which are further transformed into query representations and table representations. A graph-based prediction is obtained from node embeddings and a text-based prediction is obtained from prior methods focusing on text matching [69, 57]. Two predictions are combined to produce the final ranking scores. We find that the graph-based prediction can further improve the performance of state-of-the-art text matching method. Moreover, graph-based matching methods are more robust when some metadata information is incomplete.

Table 2.6: Similarity measures.

Measure	Equation
Early	$cos(\vec{C}_q, \vec{C}_T)$
Late-max	$max(\{cos(\vec{q}_i, \vec{t}_j\}), \vec{q}_i \in \vec{C}_q, \vec{t}_j \in \vec{C}_T$
Late-sum	$sum(\{cos(\vec{q}_i, \vec{t}_j\}), \vec{q}_i \in \vec{C}_q, \vec{t}_j \in \vec{C}_T$
Late-avg	$avg(\{cos(\vec{q}_i, \vec{t}_j\}), \vec{q}_i \in \vec{C}_q, \vec{t}_j \in \vec{C}_T$

2.5.3 Search by Example

The queries used for metadata search and tabular dataset search are keyword queries, which are constructed by users and represent their information needs. An alternative way to represent a user's information need is by providing an example dataset, so that a dataset search engine can find related datasets. We call this task *dataset recommendation* or *search by example.*

Metadata-based Search. Indexed metadata fields provide informative signals for dataset search engines since metadata is expected to describe a dataset. Datasets can have different modalities for their dataset content but metadata is in text and it allows us to use existing text matching methods for dataset recommendation. For example, the title of a dataset can be used as a keyword query to match titles of other datasets. Datasets that are owned by the same provider or have the same hashtag can also be recommended[18]. DataLab [279] obtains vector representations of datasets from their text descriptions extracted from research papers and dataset recommendation scores are calculated from vector similarities.

Table-based Search. Metadata-based search is modality-agnostic since dataset content is not used and its modality does not matter. The query of table-based

[18] `data.world`

search is a table. As mentioned by Zhang et al. [300], table-based search serves as an intermediate step for many downstream applications such as data completion and table augmentation. In fact, dataset search in general can be useful for various downstream applications.

One line of approach to table-based search focuses on table similarity prediction. Zhang et al. [301] use the method in Zhang et al. [299] to map a table into different semantic spaces. Then multiple similarity scores between a pair of table can be calculated based on Table 2.6. A random forest is used to map those similarity scores into a final table matching score. TabSim [100] learns the semantic representation of a tabular dataset from its caption and tabular content. A Bi-LSTM [231] is used to learn caption representation and a newly designed layout-aware network is used to learn tabular content representation. Then two representations are concatenated as the tabular dataset representation. The relatedness between a query table and a candidate table is measured by a Siamese neural network.

Another line of approach finds relevant tables that are joinable or unionable with the query table. Just like the join and union operation in database, those table-based methods can expand a query table with more columns or rows. The Mannheim Search Join Engine [140] provides a join operator so that additional attributes can be added to a user-provided query table. They first find candidate tables where at least one matching header of the query table is found. At the token level, they query a Lucene index to retrieve tables where at least one token in the headers matches the tokens in the query table's headers. An alternative way is to use a fuzzy matching function [262] to calculate the similarity between a query header and the matched candidate table header. Then the final matching score is computed as the average similarity between the matched headers from the query table and the candidate table.

Nargesian et al. [178] propose to solve the table union search problem which is to find target tables that have the highest likelihood of being unionable with a query table on some subset of attributes/headers. For three types of unionability, different statistical tests are designed to determine how likely a hypothesis that two attributes are unionable is to be true. The first type, named set unionability, uses the size of value overlap between two columns. The second type, called semantic unionability, assumes the headers appear in an ontology so that the semantic similarity between two headers can be estimated more accurately. The third type, which is natural-language unionability, assumes the values of attributes are part of a natural language. For this case, word embeddings pretrained on Wikipedia are used to learn representations for both attributes and values so that the likelihood of two attributes are unionable can be further estimated. The matching score between a query table and a candidate is determined by the maximum attribute unionability score.

Google Fusion Tables [71] define a related table which is either an entity complement or a schema complement of the query table. A higher entity complement score indicates that the candidate table is more likely to be unionable and a higher schema complement score indicates the candidate table is more likely to be unionable. Entity complement score is the product of entity consistency score and expansion score. The entity consistency score measures the relatedness of entity sets in the query table and a candidate table. The relatedness of a pair of entities is calculated by the dot product of their entity representations which are vectorized labels obtained from external resources. And the entity consistency score between two tables are aggregated (e.g., averaging or summation) from pairwise entity relatedness scores. The expansion score measures the attribute similarity between two tables and the pairwise attribute matching scores are aggregated to compute an overall expansion score. Schema complement score is obtained by the product of entity coverage score and attribute benefit score. Entity cover score is calculated as the fraction of entities in one table covered by another table. Attribute benefit score quantifies the benefits of adding additional attributes from a candidate table to the query table. Zhu et al. [310] speed up the candidate table retrieval stage by a new index structure called Locality Sensitive Hashing Ensemble. JOSIE [309] formalizes the joinable table search problem as an overlap set similarity search problem by considering columns as sets and matching values as intersection between sets.

Table-based search becomes more and more important for modern data science and analysis [168]. Searching over massive data repositories (also called data lakes) enables data integration in the workflow more effectively and efficiently [305].

2.6 Search as a Service: Data-centric applications

Dataset search can be considered as an interface to an extremely large knowledge base and sometimes serve as an intermediate step for downstream applications. Juneau [305] provides the function of searching related tabular datasets so that tables with similar metadata can be used as augmented training data for machine learning applications. Benefiting from the large amount of indexed data, dataset search can also be used for data imputation [6] where the missing values of a dataset can be recovered. Many applications adopt a two-stage framework where the first stage is a dataset search application. Open domain question answering systems [201, 66, 51] rely on a generic dataset search engine to find a small set of candidate tables so that more complicated models can be used for fine-grained answering.

2.7 Summary

In this chapter, we systematically discuss several aspects about a dataset search engine. We summarize different techniques to obtain datasets that can be indexed by a dataset search engine. We present multiple dataset management tasks that can help prepare datasets for downstream applications or improve the quality of datasets. Different types of dataset search tasks are also introduced. The material in this chapter can be used as a guideline for dataset search engine developers.

Chapter 3

Dataset Acquisition from Academic Literature

3.1 Introduction

In recent years, there has been a significant increase in the number of published papers for AI related tasks, and this leads to the introduction of new tasks, datasets, and methods. Despite the progress in scholarly search engines, it is challenging to connect previous technologies with new work. Researchers from the semantic web community have noticed the importance of organizing scholarly data from a large collection of papers with tools like the Computer Science Ontology [227]. Natural language processing researchers have proposed methods to extract information from research articles for better literature review [179]. Unlike prior work which focuses on the extraction of paper metadata and key insights, we design an ontology and knowledge base for better evaluation of AI research. Papers With Code[1] is a website that shows the charts of progress of machine learning models on various tasks and benchmarks. Those charts can help researchers to identify the appropriate literature related to their work, and to select appropriate baselines to compare against. Although manually updating this leaderboard may keep it accurate, it will become

[1] `https://paperswithcode.com/`

more difficult and time consuming as more and more papers are published.

Knowledge extraction from research papers has been studied by the information extraction (IE) community for years. Hou et al. [104] extract ⟨*Task*, *Dataset*, *Metric*, *Score*⟩ tuples from a paper where the paper content is extracted from pdf files. In their two-stage extraction framework, they first extract ⟨*Task*, *Dataset*, *Metric*⟩ tuples, and then for each tuple, they separately extract ⟨*Dataset*, *Metric*, *Score*⟩ tuples. Kardas et al. [118] specifically focus on extracting table results by taking advantage of available LaTeXsource code of papers. Work by Jain et al. [112] developed in parallel to ours uses data from Papers With Code as a distant supervision signal and introduces a new document-level IE dataset for extracting scientific entities from papers. Our work is complementary to AI-KG [76] which takes the abstract of a paper as input. We also consider other sections and tables in a paper where the evaluation scores of different metrics always occur. Our ontology can be considered as the front end of a knowledge system that organizes all extracted knowledge from different backend IE tasks.

In this chapter, we first introduce the Machine Learning Progress Ontology (MLPO) which defines the core entities and relations useful for progress tracking of AI literature. Then, we propose to construct the Machine Learning Progress Knowledge Base (MPKB) from a paper corpus using information extraction techniques. The ontology definition and pipeline of knowledge construction are available online[2].

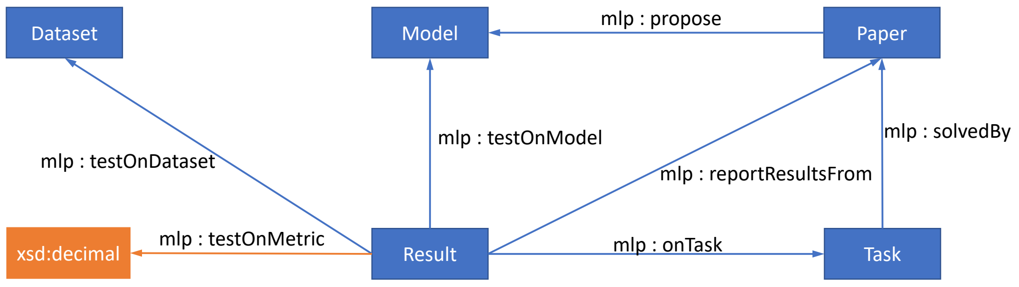
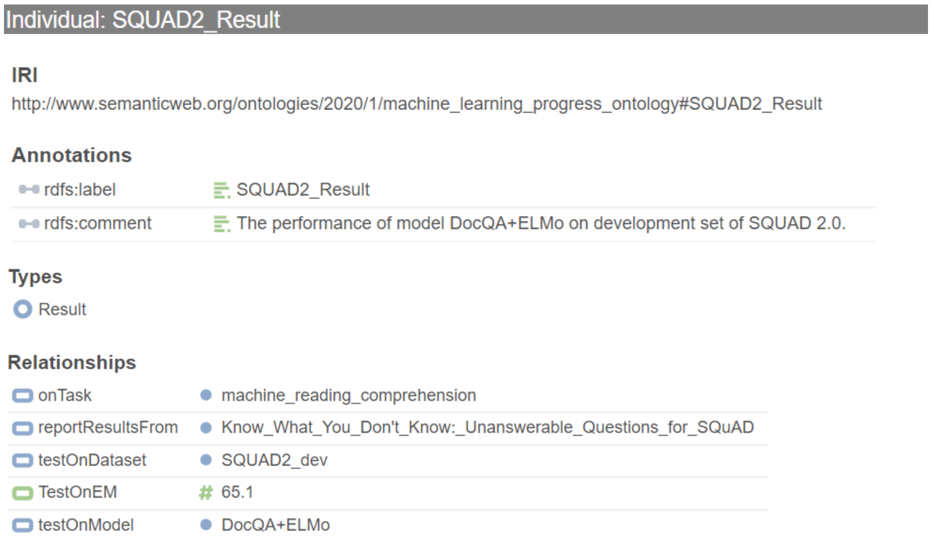


Figure 3.1: The main classes and relations in MLPO. The blue arrow signifies a object property and the orange arrow signifies a data property.

[2]https://github.com/Zhiyu-Chen/Machine-Learning-Progress-Ontology

3.2 Machine Learning Progress Ontology

As shown in Figure 3.1, the MLPO focuses on the results of machine learning experiments, which differentiates it from prior work. This ontology defines five core classes: *Task*, *Dataset*, *Result*, *Model* and *Paper*. To support proper citation of results, it also includes general properties such as Venue, Author and Title which have already been defined in the BIBO ontology[3]. In total, MLPO has 22 classes, 18 object properties and 24 data properties.


Individual: SQUAD2_Result

IRI
http://www.semanticweb.org/ontologies/2020/1/machine_learning_progress_ontology#SQUAD2_Result

Annotations

rdfs:label	SQUAD2_Result
rdfs:comment	The performance of model DocQA+ELMo on development set of SQUAD 2.0.

Types
Result

Relationships

onTask	machine_reading_comprehension
reportResultsFrom	Know_What_You_Don't_Know:_Unanswerable_Questions_for_SQuAD
testOnDataset	SQUAD2_dev
TestOnEM	65.1
testOnModel	DocQA+ELMo

Figure 3.2: An example of extracted Result individuals.

It is important to notice that the *Result* class connects to all other core classes. From a single paper, we could extract multiple *Result* individuals and each *Result* individual records the used dataset, the used model, the target task and also the reported evaluation score. For *Task* class, we create different subclasses representing different AI tasks (e.g., natural language processing task). We create various data properties for evaluation metrics which have different range constraints. For example, the range of data property "TestOnEM" which represents the exact matching

[3]`https://www.dublincore.org/specifications/bibo/`

metric, is a decimal as shown in Figure 3.2. We use WebProtégé[4] to develop our ontology and an example of extracted individuals is shown in Figure 3.2.

3.3 Knowledge Base Construction

Constructing the Machine Learning Progress Knowledge Base (MPKB) involves two tasks: scientific entity recognition (SER) and relation classification (RC). For the SER task, we identify the core entities in a paper which are datasets, tasks and metrics. For the RC task, for simplicity, here we only show how to identify two relations in a paper: whether a dataset is used for a task and evaluated with a metric. For the example in Figure 3.2, we would like to know whether "SQUAD2_dev" is used for the task of "machine_reading_comprehension" and is evaluated with "TestOnEM". We believe the methods can also be applied to recognize other entities and relations. We leave extracting all the mentioned relations defined in MLPO to future work.

3.3.1 Scientific Entity Extraction

We treat entity extraction as a sequence tagging problem. One challenge is that we only have document-level instead of sequence-level annotations. As a solution, we use fuzzy matching to find the entity spans in a paper. Given the text of a paper, we first use spaCy[5] to find the noun phrases. Then we match the noun phrases with precurated entity names using the similarity measure based on Levenshtein Distance[6]. For tasks and metrics, we set the similarity threshold to 0.5. For datasets, we set the matching threshold to 1 (i.e., exact match). If the fuzzy matching similarity between a noun phrase and an entity name is larger than the corresponding threshold, then we annotate the noun phrase as the target entity. We also designed a tagging schema similar to BILOU [215]. For section titles in the paper, we annotate every token either as at the first, middle or last position. For every sentence in each section, we tag the word as at the first, middle or last position of the sentence. For tokens

[4]`https://webprotege.stanford.edu/`

[5]`https://spacy.io/`

[6]`https://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-\matching-in-python/`

belonging to an entity in a sentence, we tag them with the corresponding entity types. Based on the paper text and annotated tags, we train a Bi-LSTM-CRF model [108] to predict the tags of test data.

3.3.2 Relation Classification

We use an information retrieval method for relation classification. To construct the query q, we concatenate the text of a result tuple $\langle$*Task*, *Dataset*, *Metric*$\rangle$. We select the first 100 tokens from each section of a paper as its text representation T_p. Finally, we match the two inputs with a neural ranking model. In particular, we use Conv-KNRM [69] to predict the binary relevance score of a triple-paper pair:

$$label = ConvKNRM(q, T_p) \tag{3.1}$$

The label is equal to 1 if the triple is relevant to the paper, otherwise the label is equal to 0. We choose Conv-KNRM in this paper because it is efficient. A state-of-the-art model like BERT [77] can also be used as in Hou et al. [104].

3.4 Experiments and Evaluation

Constructing the knowledge base for scientific datasets has three steps. First, we recognize the entities in papers. Then we build the relations among entities. In the final step, we aggregate all the extracted information for papers to form the full knowledge base. In this section, we evaluate the performance of the first two steps separately.

We randomly divided the paper collection of the **NLP-TDMS** dataset [104] into training (80%) and testing (20%) sets. For the Bi-LSTM-CRF model, we set the embedding dimension to 100. We use a bi-LSTM with 2 layers. We use Adam optimizer for gradient descent with a learning rate that is equal to 0.0001. When training relation classification, we create k positive result tuple-paper pairs (one for each tuple used to annotate the paper) and $n - k$ negative pairs, where n is the total number of result tuples in the ground truth. In the training phase of relation classification, given a set of n triples that represent the ground truth label collection, if a paper is annotated with k triples, we create k positive triple-paper pairs and $n - k$ negative pairs. Binary cross entropy loss is used to train the model. This results in many more negative samples than positive samples: 94% of result tuple-paper pairs are negative. To address this imbalance, we oversample the positive class by creating 20 copies of each positive sample. In addition to that, when creating negative samples for the collection of labels (TDM triples), there are negative triples that are very similar to the positive triples, and this can lead to confusion in the neural model when learning the parameters. In order to solve this problem, we use fuzzy matching to calculate the similarity between a given negative label and the positive labels of a paper. We discard negative samples that have similarity with positive triples larger than 50.

From the result in Table 3.1 and Table 3.2, we can see that among different entity types, *Task* is the easiest type to recognize. *Dataset* has higher precision but lower recall than *Metric*. Such variances may indicate that tasks have more observable patterns to appear in a paper than other entity types, so that the predicted sequence tagging is more accurate. Conv-KNRM achieves high results on all the evaluation metrics for predicting irrelevant paper-triple pairs. The most challenging part for the neural network is to capture the semantic similarities between paper content and $\langle$*Task*, *Dataset*, *Metric*$\rangle$ triple for a positive pair.

Table 3.1: Results of entity extraction.

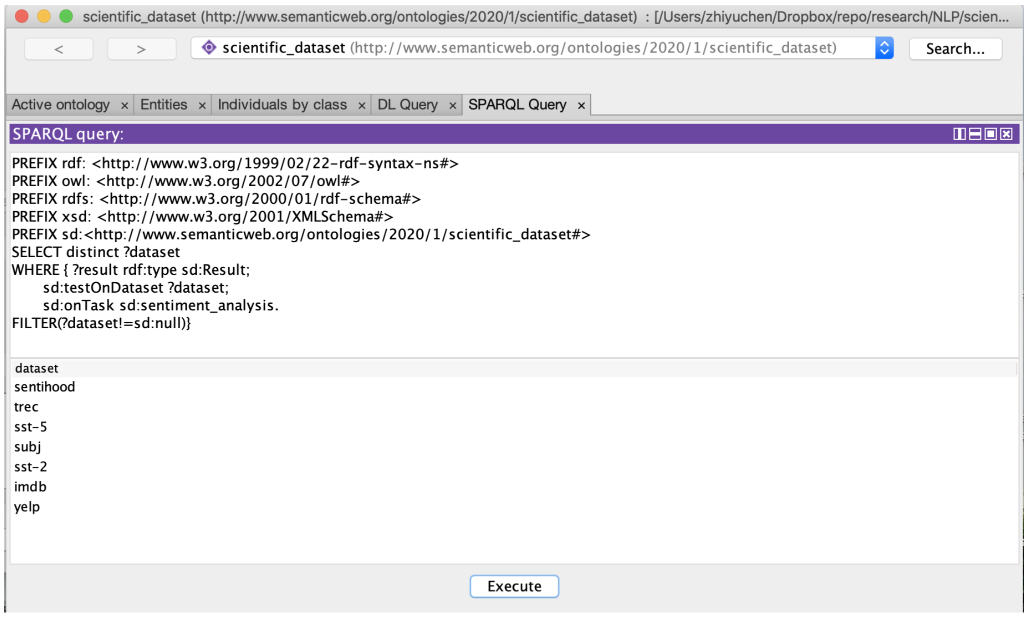
Tag	Precision	Recall	F1
Task	0.99	1.00	0.99
Dataset	0.66	0.36	0.46
Metric	0.44	0.46	0.45

Table 3.2: Results of relation classification.

Paper-triple label	Precision	Recall	F1
Irrelevant (0)	0.93	0.99	0.96
Relevant (1)	0.98	0.51	0.67

3.5 Dataset Search on KG

For each paper, we do entity extraction and relation classification. We save the results into one OWL file[7] with OWL syntax as the final knowledge base. Based on the knowledge base, we are able to answer some queries. For example, we can query the datasets that are used in a given task. We test our knowledge base using Protege[8]. In Figure 3.3, we show a use case where we ask the knowledge base what are the datasets for the task of sentiment analysis.



scientific_dataset (http://www.semanticweb.org/ontologies/2020/1/scientific_dataset) : [/Users/zhiyuchen/Dropbox/repo/research/NLP/scien...
scientific_dataset (http://www.semanticweb.org/ontologies/2020/1/scientific_dataset)
Search...
Active ontology
Entities
Individuals by class
DL Query
SPARQL Query
SPARQL query:
Execute

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sd:<http://www.semanticweb.org/ontologies/2020/1/scientific_dataset#>
SELECT distinct ?dataset
WHERE { ?result rdf:type sd:Result;
      sd:testOnDataset ?dataset;
      sd:onTask sd:sentiment_analysis.
FILTER(?dataset!=sd:null)}
```

```
dataset
sentihood
trec
sst-5
subj
sst-2
imdb
yelp
```

Figure 3.3: Demonstration of querying the constructed knowledge base.

[7]`https://www.w3.org/2007/OWL/wiki/Document_Overview`
[8]https://protege.stanford.edu/

3.6 Summary

In this chapter, we have proposed an ontology specifically designed for progress tracking of AI tasks such as natural language processing and computer vision. We first have defined the core entities and relations in the proposed Machine Learning Progress Ontology (MLPO). Then we have proposed methods to extract information from papers to construct a knowledge base for AI evaluation. The resulting knowledge graph can be used for various downstream tasks. In the end of this chapter, we show an example of how to search over the constructed KG through a SPARQL query.

3.7 Bibliographic Notes

In this section, we review three lines of relevant research work: (1) information extraction for scientific papers; (2) natural language inference. We discuss how they are related to our work.

There is some related work from the information extraction community. Gábor et al. [90] propose unsupervised methods to annotate concepts and semantic relations in paper abstracts. The task was then extended to SemEval-2018 Task 7 [89]. Luan et al. [155] extend the dataset by adding more relation types. They also develop a framework called Scientific Information Extractor to extract six types of scientific entities and seven relation types. However, all such efforts only focus on paper abstracts and information in other sections of a paper is not considered. Besides, there is a gap from information extraction to knowledge base construction. For example, a paper can mention multiple tasks and datasets, but it is possible that not all tasks or datasets are studied in the paper, which means the extracted positive entities and relations are negative cases under the context of constructing a knowledge base that records the tasks and datasets studied in a paper. The closest idea to our work was proposed in Hou et al. [104], where the authors extract ⟨Task, Dataset, Metric, Score⟩ tuples from a paper where the paper content is also used. In their two-stage extraction framework, they first extract ⟨Task,Dataset,Metric⟩

tuples (TDM tuples), and then for each tuple, they separately extract ⟨Dataset, Metric, Score⟩ tuples (DMS tuples). Both stages are considered as a natural language inference task where the premise comes from the paper and the hypothesis is a targeted tuple. The weakness of their method is that their hypotheses (TDM or DMS tuples) are predefined which means they skip the named entity recognition step. For a new paper, there could be a large number of target tuples to be classified and it is unnecessary and ineffective to classify all the hypotheses. However, this is an important step in the knowledge base construction pipeline. Due to lack of annotated data, we use their annotations as a starting point and also "recover" the complete pipeline of constructing a knowledge base with the named entity recognition as the 1st step.

The task of natural language inference, also called recognizing textual entailment, is to determine whether a "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given a "premise". In general, it involves classifying the relationship between two sentences. The release of the Stanford Natural Language Inference dataset [30] made it possible to achieve good results on this task with neural network methods. More recently, BERT [77] and several other pretrained models have achieved the new state-of-the-art results on multiple NLI benchmarks[9].

In this chapter, we also formulate the relation classification task as a natural language inference task. Given two sentences including different target entities, we classify the relationship of the two entities (sentences).

[9] `https://gluebenchmark.com/leaderboard`

Chapter 4

Schema Label Management for Datasets

Impoverished descriptions and convoluted schema labels are common challenges in data-centric tasks such as schema matching and data linking, especially when datasets can span domains. To address these issues, we consider the task of schema label generation. Typically, schema labels are created by dataset providers and are useful for users to understand a dataset. The motivation behind the task is that a lot of data linking systems require overlapping information between two datasets and rely on unique identifiers of schema labels. Moreover, it is common for schema labels in different datasets to have different identifiers even when they refer to the same concept. With no naming standard for schema labels, unintelligible labels are widely found in real-world datasets. For example, many schema labels contain abbreviations and compound nouns that hinder automated matching of attributes in corresponding datasets. Through schema label generation, more common (and thus understandable) schema labels can be provided to allow for broader schema matches in contexts such as dataset search and data linking. We develop a variety of features based on analysis of dataset content to enable machine learning methods to recommend useful labels. We test our approach on two real-world data collections and demonstrate that our method is able to outperform the alternative approach.

4.1 Introduction

Organizations and individuals worldwide make datasets public and enable users to freely explore such valuable resources. An increasing number of online data sources, such as governmental data portals (e.g., data.gov[1], data.gov.uk[2] and data.gov.sg[3]), and more general facilities like datahub[4] (alongside older sites such as the UCI machine learning repository[5]), suit the diverse needs of data experts, researchers, and journalists. More data also means more challenges as it becomes a non-trivial task to effectively integrate datasets from different resources. In order to help agencies to manage their data, the U.S. government released a policy[6] to instruct (government) data providers to provide metadata with their datasets. It is a great opportunity for technical communities to bring heterogeneous data together for diverse applications and also a big challenge which may require advanced approaches to manage the datasets. However, many datasets do not adopt metadata standards so that open source data management systems (e.g., CKAN[7]) are unable to be utilized. Another challenge is that different agencies are likely to have different data formats and standards [254] resulting in difficulties in merging heterogeneous datasets.

Among all types of data, tabular data or the data table is one of the most important. It presents relational data in a compact way and is commonly used in different applications such as knowledge management and web data presentation. A data table usually has a header row, consisting of schema labels (attribute names), followed by data rows storing the actual data values of corresponding attributes. In this chapter, we focus on this simple data table format although there are data tables with more complex structures where headers are nested. Tabular data is widely used in different communities because it clearly shows the relationships of different entities and facilitates data analysis. Many tools, such as Microsoft Excel,

[1]`https://www.data.gov/`
[2]`https://data.gov.uk/`
[3]`https://data.gov.sg/`
[4]`http://datahub.io/`
[5]`http://archive.ics.uci.edu/ml/index.php`
[6]`https://digital.gov/open-data-policy-m-13-13/`
[7]`https://ckan.org`

Google Spreadsheets and Tableau, can easily work on tabular data for analysis and visualization.

Current data linking systems usually rely on the overlapping information in data itself or more commonly, the corresponding metadata fields such as title, tags, description and publisher. However, non-dictionary words (NDWs) commonly appear in data tables and can have a significant impact on data linking. The existence of NDWs in schema labels is also a well-known problem in schema matching systems [214, 236]. However, users are less likely to use query terms which are NDWs so that datasets with NDWs as headers are less likely to be returned by the retrieval system. The case is even worse if the dataset has neither table headers consisting of dictionary words nor text description.

Table 4.1: Sample from a dataset of NYC Farmers Markets.

Farmers Market Name	...	State	Zip Code	Latitude	Longitude
Carroll Gardens Greenmarket	...	NY	11231	41	-74
Cortelyou Greenmarket	...	NY	11226	41	-74
Cypress Hills Youthmarket	...	NY	11208	41	-74
East New York Farm Stand	...	NY	11207	41	-74
East New York Farmers' Market	...	NY	11207	41	-74
Fort Greene Park Greenmarket	...	NY	11205	41	-74
...	...	...	...	...	...
Graham Avenue Farmers' Market	...	NY	11206	41	-74

Table 4.2: Sample from an Oceanographic dataset.

cruiseid	year	si	month_gmt	day_gmt	time_gmt	...	lat	lon
EN319	1999	T.Durbin	2	21	29.3	...	41.4922	-71.4187
EN323	1999	J.Ledwell	5	14	1146.88	...	41.5234	-70.6723
EN330	1999	C.Greene	10	23	140.4	...	42.5035	-66.8025
OC342	1999	B.Houghton	5	24	19.5	...	41.0683	-67.4617
...	...	...	...	...	...	...	...	...
OC343	1999	D.Hebert	6	25	731.47	...	40.9997	-67.6014

To address this problem, we propose a supervised method which recommends alternative schema labels. Considering Tables 4.1 and 4.2 from different domains, we can easily identify that the column "latitude" in Table 4.1 refers to the same concept as "lat" in Table 4.2, but it is not an easy task for data linking and schema matching systems. If we can recommend the column "lat" with new schema labels such as "latitude", "location", it can not only facilitate integrating the column of Table 4.2 with other columns, but also help users to better understand the meaning of this column.

We construct a variety of features from column content and enable machine learning models to generate alternative schema labels. To evaluate our method, we test on datasets with different heterogeneity and show that the features are effective for the schema label prediction task. Additionally, we experiment on integer columns, float columns, string columns and show that our method provides consistent performance on those different columns types.

In this chapter, we summarize our contributions as follows:

- We propose a domain-independent method for schema label prediction.
- We run experiments on real world datasets with varying degrees of heterogeneity and demonstrate the effectiveness of our methods on the task of schema label prediction. Our experimental results suggest that the difficulty of the task increases with the heterogeneity of the datasets.
- We evaluate our method on columns of three basic data types (integers, floats, and strings) and demonstrate that our method outperforms the baseline on each of them.

4.2 Methods

4.2.1 Problem Statement

In this chapter, we focus on finding alternative schema labels (column names) based on analysis of the content of the column.

We consider data tables with n columns and $m + 1$ rows with the following format:

$$\begin{bmatrix} c_1 & c_2 & ... & c_n \\ v_{1,1} & v_{1,2} & ... & v_{1,n} \\ ... & ... & ... & ... \\ v_{m,1} & v_{m,2} & ... & v_{m,n} \end{bmatrix}$$

For convenience, we use the following naming conventions in the rest of the chapter:

- **schema label** (or column name): c_j, where $j \in [1, ..., n]$.
- **schema content** (or column content): $C_j = \{v_{1,j}, ..., v_{m,j}\}$, $j \in [1, ..., n]$
- **column**: $(c_j, C_j), j \in [1, ..., n]$.

Given C_j and k target labels $L = \{l_1, l_2, ..., l_k\}$, our objective is to learn a function that models $P(l|f(C_j))$, $(l \in L)$ where f is a function extracting features from C_j. The features will be introduced in the following section. A perfect prediction should satisfy:

$$c_j = \arg\max_{l \in L} P(l|f(C_j))$$

4.2.2 Schema Label Prediction Features

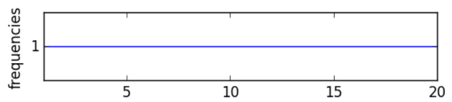
Our approach to predicting schema labels is to leverage features extracted from column content. Past research has indicated that useful features are important for table understanding [271, 110]. We assume that a schema label and evidence observed from the column content are highly related. One obvious example is that column contents corresponding to different data types are significantly different. Though column data types are not provided directly for the majority of public datasets, machine learning models are able to identify such features automatically [255].

Our task is much more challenging than just inferring the data type, since the number of data types is small and constant while the number of possible schema labels is uncertain but large. It also means our model should be able to capture the differences among columns with the same data type. As shown in Table 4.1, a column of zip codes usually have data cells consisting of five digits, while a column of latitudes usually have data cells that are real numbers ranging from -90 to 90. If there is a column in the data table without a header and we know all the values in the column are five-digit numbers, the header is more likely to be "Zip Code" instead of "latitude". Therefore, for possible numerical columns, the *maximum value* and *minimal value* are important features to characterize them. However, not all columns are numerical columns. Then for non-numerical columns, we instead use the average maximum value and average minimal value of other columns in order to appropriately minimize the impact of these features.

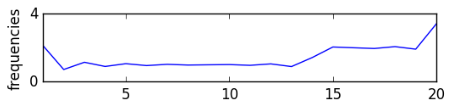
We define *content unique ratio* and *content histogram* to describe the distribution of cell values. Content ratio [82] is usually used as a feature to categorize the class of table where the ratio of cells containing content of a specific type is calculated. Similarly, we use content unique ratio to categorize a column where the proportion of the number of unique cells over the number of all cells is calculated. In Table 4.1, the content unique ratio is $1/102 \approx 0.01$ for "State" if the table has 102 rows and all cell values under this schema label are all "NY". In contrast, the content unique ratio is $102/102 = 1$ for "Farmers Market Name" if all cell values under this schema label are different.

A content histogram contains more fine-grained information about the content distribution than the content unique ratio. To obtain the content histogram, we rank the unique cell values by frequencies (low-frequency first) and generate a vector where the i^{th} dimension is the frequency of the i^{th} ranked cell value. For different column contents, we could obtain the vectors of various lengths. For data.gov and WikiTables which are two datasets used in our experiment, the medians are 26 and 13 respectively. Therefore, we generate the content histogram by resampling the vector to a 20-dimensional vector using FFT transformations[8]. We show the content histogram of "Farmers Market Name" and "Zip Code" of Table 4.1 in Figures 4.1a and 4.1b, respectively. The flatter shape of estimated frequencies of "Farmers

[8]We use the method from `https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.signal.resample.html`



(a) 20-dimensional content histogram of "Farmers Market Name" in Table 4.1.



(b) 20-dimensional content histogram of "Zip Code" in Table 4.1.

Figure 4.1: Examples of content histograms.

Market Name" indicates the content distribution is closer to a uniform distribution than "Zip Code".

If we treat each column content as a document, then the schema label prediction can be seen as a document classification task, where classes are possible schema labels. So it is reasonable to incorporate bag-of-words (BoWs) representation as features. For column c, we construct the *BoWs features* as

$$B_c = \{freq(u_1), ..., freq(u_i), ..., freq(u_n)\},$$

where n is the vocabulary size, u_i is the i^{th} word in the vocabulary and $freq$ represents the function calculating the frequency of u_i in c. To save memory, we only use character-level unigrams in BoWs (e.g., "EN319" is decomposed into "E","N","3","1" and "9"). In our experiment, we use BoWs features to construct the baseline method. The difference is that instead of considering character-level unigrams, we use the TF-IDF representation of tokens extracted from column content.

In a study of table header detection [86], Fang et al. showed that single row features could differentiate header rows and data rows. Inspired by their work, we extract the following *single column features* on each column instead of each row: number of characters, percentage of numeric characters, percentage of alphabetic characters, percentage of symbolic characters, percentage of numeric cells, average cell length, maximum cell length and minimum cell length. These features could be considered as an extension of the *BoWs features* which summarize the statistics of

BoWs.

We summarize all the features in Table 4.3.

Table 4.3: A list of curated features for schema label prediction

ID	**Feature length**	**Description**
1	1	maximum value in the column content
2	1	minimum value in the column content
3	1	content unique ratio
4	20	content histogram
5	# of unique unigrams[9]	BoWs (character-level unigram)
6	1	number of characters
7	1	percentage of numeric characters
8	1	percentage of alphabetic characters
9	1	percentage of symbolic characters
10	1	percentage of numeric cells
11	1	average cell length
12	1	maximum cell length
13	1	minimum cell length

4.3 Experiments

In this section, we first discuss the datasets used in our experiments. Then we evaluate our method from two perspectives. In exact schema label prediction and normalized schema label prediction, we evaluate performance of the model and demonstrate the usefulness of the aforementioned features.

4.3.1 Datasets

For our first dataset, we collected all available comma-separated value (CSV) files (7485 in good format) from Data.gov which are contributed by more than 50 U.S. government agencies. This dataset covers a variety of topics such as agriculture, climate, economy, and health. Web tables are also tabular tables and have an important role in applications like Web Data search and knowledge base construction.

[9] 741 for data.Gov and 54982 for WikiTables.

Therefore, we also experiment on WikiTables [27], which contains 1.6M tables extracted from Wikipedia.

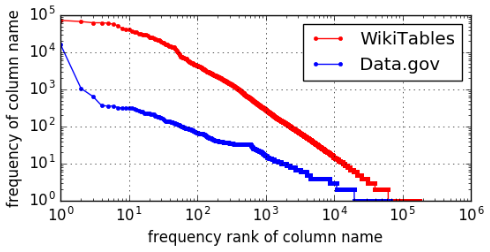


Figure 4.2: Rank-sorted frequencies of column labels

We observe that raw schema labels exhibit properties similar to terms in natural language, in that the rank-sorted frequencies of schema labels produce a curve which approximates the well-known Zipf's law and reflects the heterogeneity of schema labels.

4.3.2 Exact schema label prediction

We first evaluate our method on the task of exact schema label prediction. Considering a collection of tabular datasets where many schema labels are blank, our goal is to predict the missing schema labels by the corresponding column content. Specifically, we consider two questions: 1) How useful are features extracted from dataset content for schema label prediction? and 2) Does heterogeneity of the dataset collection make the task more difficult?

To answer the first question, we build machine learning models using the features proposed in Section 4.2.2, and evaluate the prediction results under different metrics. We treat schema label prediction as a multiclass classification task, where each schema label in the training set represents a class. We calculate macro-averaged and micro-averaged precision, recall and F-score of predictions on the test set. Macro-average is the mean of scores of all the classes, thus giving equal weight to each class. Micro-average, giving equal weight to each prediction decision, is the score obtained by globally counting the total true positives, false negatives and false positives.

Larger classes have a larger contribution to the micro-average. In the multiclass classification scenario, the micro-average precision, recall and F-score are the same, thus we only show the Micro F-score in our results. We also report the top-n accuracy which is the fraction of test data for which the correct label is among the top-n labels considered most probable by the model.

The second question can be implicitly answered by comparing the results of the following datasets:

- **Gov_Rand**: 300 datasets are randomly selected from Data.gov.
- **Gov_NY**: 300 datasets are randomly selected from Data.gov published by NYC Open Data[10].
- **Wiki_Rand**: We experiment on 554218 tables from WikiTables which have at least 4 columns and 6 rows. Since a lot of tables are in unexpected format, we further filter those columns whose schema labels appear no more than 100 times.

The sizes of each dataset and training and testing partitions are found in Table 4.4.

Different data owners usually publish datasets in different domains, with different vocabularies and thus have different pattern of schema label creation. Thus, the difficulty caused by heterogeneity can also be shown by comparing the results on Gov_Rand and Gov_NY. Since there are only 327 datasets published by NYC Open Data, we randomly select 300 datasets for both Gov_Rand and Gov_NY so that the results from the models are more comparable.

For our experiment, we train random forest classifiers using the curated features introduced in Section 4.2.2. The default parameters of scikit-learn implementation[11] are used except that the number of trees in the forest is set up to 25 (to reduce memory requirements).

[10]`https://opendata.cityofnewyork.us/`

[11]`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

Table 4.4: Statistics of extracted columns

Dataset	#train	#test	#classes
Gov_Rand	3833	1644	4048
Gov_Rand (freq >1)	1415	607	593
Gov_NY	2799	1200	2494
Gov_NY (freq >1)	1391	597	483
Wiki_Rand	806755	1882425	2234

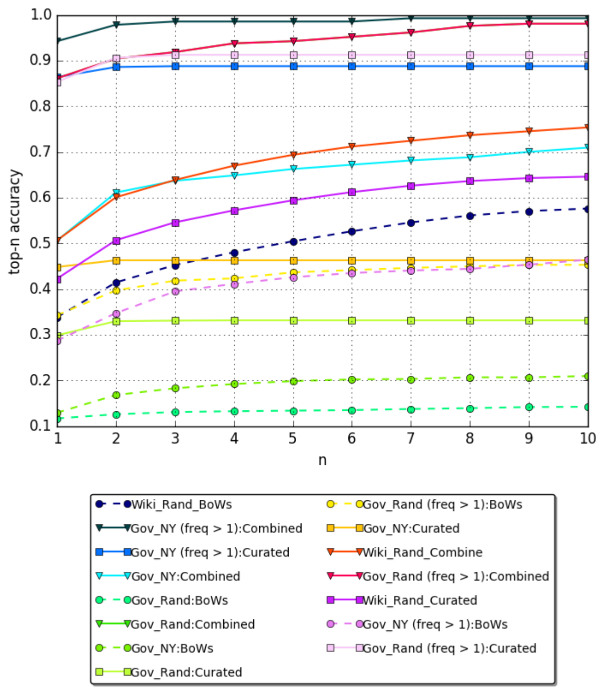


Figure 4.3: Top-n accuracy of exact schema label prediction

As a **baseline**, we use the same classifier setting training on TF-IDF features extracted from column contents where each cell is tokenized.[12] It is important to notice that a large number of numeric values appear in the datasets which result in an extremely large vocabulary size. Thus dimension reduction is helpful in order to improve the classification efficiency. Truncated SVD[13] (a.k.a., LSA) is used to reduce

[12]`http://www.nltk.org/_modules/nltk/tokenize/toktok.html`

[13]`http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html`

Table 4.5: Micro average and Macro-average scores of exact schema label prediction

Features	**Dataset**	**Micro-F**	**Macro-P**	**Macro-R**	**Macro-F**
Curated	Wiki_Rand	0.42	0.30	0.21	0.23
	Gov_Rand (freq >1)	0.85	0.79	0.83	0.80
	Gov_Rand	0.30	0.13	0.15	0.14
	Gov_NY (freq >1)	0.86	0.79	0.83	0.80
	Gov_NY	0.45	0.18	0.21	0.19
BoWs	Wiki_Rand	0.34	0.34	0.16	0.19
	Gov_Rand (freq >1)	0.35	0.23	0.27	0.23
	Gov_Rand	0.11	0.05	0.06	0.05
	Gov_NY (freq >1)	0.28	0.12	0.14	0.12
	Gov_NY	0.13	0.03	0.04	0.03
Combined (Curated+ BoWs)	Wiki_Rand	0.43	0.30	0.22	0.24
	Gov_Rand (freq >1)	0.86	0.84	0.83	0.82
	Gov_Rand	0.37	0.24	0.25	0.24
	Gov_NY (freq >1)	0.94	0.82	0.85	0.83
	Gov_NY	0.49	0.31	0.32	0.30

the dimensionality of the TF-IDF representation and BoWs features to 300. We also concatenate the curated features with baseline BoWs features in order to see whether the combination of features could further improve the results. For Gov_Rand and Gov_NY, we split each of them into 70% training set and 30% testing set. Since the number of classes in Gov_Rand and Gov_NY is very large but the dataset size is relatively small, the results could be significantly affected by infrequent schema labels, especially those that only appear once. As a result, many labels in the testing set do not appear in the training set. Therefore, we also experiment on Gov_Rand and Gov_NY after filtering those columns whose schema label only appears once and we call them Gov_Rand(freq >1) and Gov_NY(freq >1) respectively.

Experimental results are reported in Table 4.5. We observe that our curated features approach achieves better results than the baseline on all datasets. For both methods, scores on Gov_NY are higher than scores on Gov_Rand, which suggests the heterogeneity caused by data creators indeed increases the difficulty of the task. After filtering those schema labels that only appear once, the scores of both methods significantly increase, since the cases in which a class in the testing set does not appear in the training set have decreased. However, our method has a more considerable degree of improvement than the baseline and achieves an F-score greater than 0.8 for both the Macro-average and Micro-average. This indicates that our method has decent performance on popular schema labels. When applied to WikiTables, we notice that the gap of performance between our method and baseline is narrower. It is likely that schema label prediction is more like a text classification problem with regard to WikiTables. Compared to datasets on data.gov, WikiTables have tabular data with smaller size, since tables with thousands of rows can hardly be displayed on a web page. A lot of datasets on data.gov are statistics and it is very likely that the content of a single column is occupied by numbers. As content extracted from an encyclopedia, WikiTables have more text description about entities and thus the content of a column is closer to a document.

Figure 4.3 shows the top-n accuracy results. The performance on datasets from data.gov has no improvement when n is bigger than 3. As we discussed before, many of the labels in the testing set do not appear in the training set, and this sets an upper bound on the accuracy of exact schema label matching. For example, there are 594 columns whose labels only appear in the testing set of Gov_NY; therefore the accuracy can never exceed $(1200 - 594)/1200 = 50.5\%$. While for Wiki_Rand, the accuracy increases when n increases for both methods.

We calculated the Gini importance of curated features of model trained on Gov_NY. For BoWs features and content histogram, we simply sum the importance scores of all the dimensions. As a result, BoWs features and content histogram make the most contribution. If we consider each dimension as a single feature, then the most important three features are total number of characters, content unique ratio and the first dimension of content histogram.

From the above observations, we know that our method outperforms the baseline in all cases. Moreover, combining our method with baseline features can further improve the performance as expected. Since we only use character-level unigram features in our method and adding word-level TF-IDF features could relieve this weakness. The prediction results can be significantly improved by filtering infrequent labels which means our methods can efficiently predict the schema labels especially for popular ones. We also confirm that the difficulty of the task increases with the heterogeneity of the datasets.

4.3.3 Normalized schema label prediction

Exact schema label prediction is a pretty strict evaluation, since a true-positive requires the predicted schema label to perfectly match the original label of the tested column content. However, there are thousands of classes and the distribution is imbalanced as shown in Figure 4.2. It is possible that a class in the training set may not appear in the testing set. However, a "wrong" prediction could be potentially useful if it refers to the same concept. For example, consider the target label "Nationality": a semantically correct prediction from the model could be "Country". Therefore, we should not consider "Country" as a wrong prediction since they refer to the same concept. More examples are shown in Table 4.6.

Table 4.6: Examples of "wrong" predictions

Original labels	**Predictions**
Year	Season
Opponent	Team
Pos	Position
Score in the final	Score

In order to relieve the situation, we first do case-folding on schema labels and then rank them by their frequencies in Gov_Rand and Wiki_Rand. From the top 2000 schema labels, we normalize a label by another label in this set which is a synonym of the original label and more human readable. In addition, 89 labels annotated as uninterpretable are removed.

Similar to Section 4.3.2, we train separate models based on different features and datasets. The results of normalized schema label prediction on Gov_Rand and Wiki_Rand are reported in Figure 4.4 and Table 4.7. As expected, the scores under different metrics significantly increase. Besides, we notice that the top-n accuracy still grows when n is bigger than 3 on Gov_Rand, which is different from exact schema label prediction. It further indicates that our model can capture the relation between a schema label and its content.

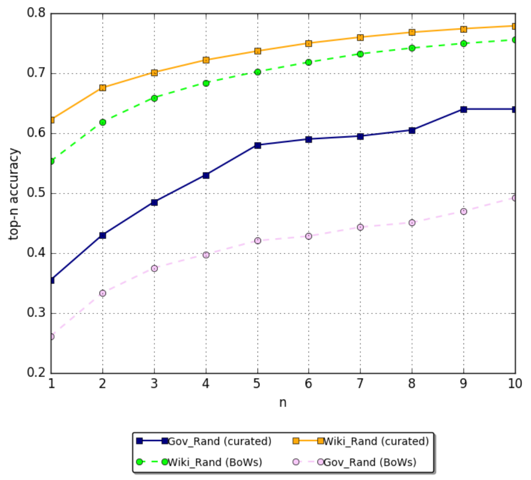


Figure 4.4: Top-n Accuracy of normalized schema label prediction

Table 4.7: Micro average and Macro-average scores of normalized schema label prediction

Features	Dataset	MicroF	MacroP	MacroR	MacroF
curated	Gov_Rand	0.36	0.21	0.22	0.20
	Wiki_Rand	0.62	0.33	0.29	0.30
BoWs	Gov_Rand	0.25	0.16	0.17	0.15
	Wiki_Rand	0.55	0.29	0.20	0.23

4.3.4 Evaluation on different data types

In this section, we evaluate schema label prediction methods on columns with different data types. We use pandas[14] to automatically infer the data type of a column and only keep those columns whose data types can successfully be identified by the IO tool. 1000 columns are randomly selected for integer type, float type and string type respectively. For each type of column, we train a model on 70% of data and

[14]https://pandas.pydata.org/

evaluate on the rest. The experiment results for our method and baseline are reported in Table 4.8. As expected, our method outperforms the baseline on all the three types of columns. It is also interesting to notice that both methods perform the best on float columns while perform the worst on string columns. There could be two reasons. First, string columns have more unique labels than float columns and integer columns, which means predicting schema labels of string columns is inherently a more difficult task. Second, some of the features are based on the numeric values in the column content, while for string columns, such features are treated as missing values and calculated from average values from other columns. Such features could be useless for string columns and damage the performance of the model. This fact indicates that designing different features for different data types could further improve the performance of schema label prediction.

Table 4.8: Micro average and Macro-average scores on different data types

Features	**Data type**	**MicroF**	**MacroP**	**MacroR**	**MacroF**
curated	integer	0.31	0.11	0.12	0.11
	float	0.37	0.10	0.11	0.10
	string	0.23	0.11	0.12	0.10
BoWs	integer	0.25	0.10	0.11	0.10
	float	0.32	0.07	0.09	0.07
	string	0.20	0.08	0.09	0.08

4.4 Summary

In this chapter, we have considered the problem of schema label prediction based on the content of a column. We treat it as a multi-class classification task, in which each class represents a schema label. A variety of features are developed to solve the problem. Our method has been evaluated on two real-world datasets: tabular data collected from data.gov and WikiTables extracted from Wikipedia. We first evaluate the approach on exact schema label prediction, which requires the predicted label to exactly match the original schema label. In this task, our method clearly outperforms the baseline on all datasets. We find that heterogeneity of the datasets likely makes the task more difficult.

Since the distribution of schema labels is quite imbalanced, many labels used in testing do not appear in the training set, which makes it inherently difficult to perform well. Therefore, we also experiment on top-ranked normalized schema labels. We select the most frequent 2000 schema labels across both datasets and merge labels that are synonyms. As expected, the scores under different metrics significantly increase compared with the results of exact schema label prediction. Additionally, we demonstrate that our method outperforms the baseline on columns of different data types. We notice that both methods perform the best on float columns while the worst on string columns, since some proposed features could be useless for string columns. It reminds us that using different features for different data types are necessary for schema label prediction.

One limitation of our current method is that we only consider the features from a single column, without considering its relationship with other columns co-occurring in the data table. Intuitively, if two columns are similar, then our method may give them the same schema label. However, it is unlikely for two identical columns to appear in the same data table. By considering the occurrence of other schema labels, such cases could be disambiguated.

One application of our method is to facilitate dataset retrieval. An existing challenge of dataset retrieval is that user queries seldom contain terms that are broadly used in schema labels, which results in low recall of related datasets. In our experiment, we find that our method often gives a prediction that are synonyms of original schema label or share the hypernym with the original schema label. Usually, the composition of NDWs schema labels is irregular and complicated but their synonyms or hyponyms could be searched by users. For example, for a column whose schema label is “Pos”, our method could predict the schema label as “Position”. However, “Position” is a term more preferred by users and more valuable to be indexed. We expect that using the predicted labels as possible term expansions (either for queries or at indexing time), the dataset retrieval system can have improved recall.

4.5 Bibliographic Notes

Related work in this chapter is in three topics: schema matching, data linking and semantic table interpretation.

In the database community, schema matching is a critical problem for integrating heterogeneous data sources, which aims to find pairwise-attribute correspondence in different schemas. It is similar to our task, except that they do not require that the pair of schema labels are exactly the same. According to the classification of Rahm and Bernstein [209], there are two major types of schema matchers: schema-only matchers and instance-based matchers. Schema-only matchers are limited to schema information such as schema name, description and data type. For example, Sorrentino et al. [237] develop a lexical annotation technique to help identify similar schema labels. However, the result of lexical annotation is strongly affected by the presence of non-dictionary words in schema labels such as abbreviations and compound words. For this reason, they expand abbreviations with the help of an online dictionary and enrich WordNet with meanings of compound nouns. The output of their system can be used as the input of another schema matching system and improve the performance of schema matching. Ratinov and Gudes [214] solve the abbreviation issue by manually designing abbreviation patterns and reduce it to a supervised pattern classification problem. As we can see, the quality of schema labels have a huge impact on schema-only matchers and thus those methods put effort into the analysis of abbreviations and compound nouns in schema labels. Even for well-known schema-only matchers, such as Artemis [40], Cupid [160] and COMA [80], they require a specified external dictionary to measure the similarity of schema labels at some steps. In real world datasets, abbreviation and compound nouns cannot cover all the complex patterns of schema labels. Sometimes the column name of a data table has no real meaning or is even missing. Moreover, available schema information of real world datasets is limited.

Our method is closer to instance-based methods where we give insight into the data content. Since we train a supervised model to use a set of existing schema labels to annotate other schema labels, the results are less sensitive to NDWs. Besides, the similarity between two schema labels relies on the similarity of corresponding content rather than the surface form of the label text. Automatch [24] uses machine learning techniques to automate the schema matching process. Their model acquires probabilistic knowledge stored in an attribute dictionary which characterizes different attributes by a set of possible values and their probability estimates. Actually, this method is similar to our baseline method which characterizes a column by its bag-of-Words representation. As mentioned by the authors, there could be endless possible values for a column, especially for those whose data types are continuous variables. Therefore, instead of considering each column value as a feature, we consider each character of each column value as a feature. We also explore other higher level features from column content and better characterize different schema labels.

In recent decades, a large number of datasets have been published in different data repositories and it becomes an infeasible task to manually link different datasets. Under this context, data linking has become an important task which aims to automatically interlink datasets and facilitate their reuse. Nikolov et al. [182, 183] present a keyword-based method with two main steps. In the first step, they use a subset of labels in the dataset as keywords to search for potentially relevant entities in external data sources. In the second step, they filter out irrelevant datasets by measuring semantic similarities used in ontology matching techniques. Leme et al. [141] propose a probabilistic method for Linked Data datasets. For a set of known datasets, they first construct a directed graph from the metadata to describe their connections. Then given a new dataset, they rank those datasets given a rank score function. A similar graph-based method is proposed by Lopes et al. [154] which treats dataset linking as a link prediction problem in social network. Ellefi et al. [22] propose a recommendation approach for data linking. They adopt the notion of dataset profiles, where a dataset is characterized as its textual descriptions and a set of schema labels. Therefore, given a source dataset, a cluster of comparable datasets can be retrieved based on their semantic similarities to a source dataset and each dataset can be ranked by tf-idf cosine similarity. A similar approach is proposed in Ben Ellefi et al. [21], where a topic-dataset bipartite graph is produced during the topic modeling process; thus a dataset can be represented as a set of topics and a topic can be modeled as a set of significant datasets. Therefore a candidate dataset can be interlinked based on connectivity within the topic-profiles graph. In de Oliveira et al. [73], the authors propose a user feedback-based approach to incrementally identify new datasets for domain-specific linked data applications. They first filter datasets according to the application queries and then use user feedback to analyze the relevance of candidate datasets. As pointed out by Nikolov et al. [182], finding the degree of overlap among datasets is critical for data linking. Schema label prediction can be a potential solution to increase the connectivity of heterogeneous datasets by recommending a dataset with schema labels that have appeared in other datasets.

As embedded data on web pages, Web tables take an important role in applications like knowledge base construction [304, 232, 220] and question answering [148, 200]. Therefore, it becomes crucial to recover semantics of Web tables. There are three main tasks in semantic table interpretation [307]: 1) annotate columns in a table with semantic concepts; 2) identify the semantic relations between columns; and 3) cell disambiguation by linking them to entities in a knowledge base. Among the three tasks, the first task is the closest to our work. TableMiner [307] uses features from context inside and outside of the table to help annotate columns containing entity mentions. Venetis et al. [258] leverage a database to attach a class label to a column if a sufficient number of the values in the column are identified with the corresponding label in the database. Wang et al. [263] use Probase to annotate a Table with related concepts. Similarly, a large number of works [245, 175, 174] also make use of knowledge bases to interpret Web Tables.

Different from Web tables, real-world datasets usually do not have enough context such as surrounding paragraphs or semantic markups which are often inserted in the web page. Moreover, there are few entities that can be linked to a knowledge base since the concepts contained in a dataset are usually too narrow (e.g., street names on a map) or too broad. The method proposed in this chapter only uses generic features extracted from the datasets and therefore only annotates columns with labels from the datasets rather than concepts from other resources.

Chapter 5

Schema Label Generation Enhanced Search for Datasets

In Chapter 4, we proposed a feature-based method to generate schema labels for datasets. In this chapter, we investigate how generated schema labels can be used in dataset search. We first propose a novel schema label generation model which generates possible schema labels based on dataset table content. We incorporate the generated schema labels into a mixed ranking model which not only considers the relevance between the query and dataset metadata but also the similarity between the query and generated schema labels. To evaluate our method on real-world datasets, we create a new benchmark specifically for the dataset retrieval task. Experiments show that our approach can effectively improve the precision and NDCG scores of the dataset retrieval task compared with baseline methods. We also test on a collection of Wikipedia tables to show that the features generated from schema labels can improve the unsupervised and supervised web table retrieval task as well.

5.1 Introduction

Dataset retrieval is receiving more attention as people from different fields and domains start to rely on datasets for their work. There are many data portals with the purpose of effective and efficient data management and data sharing, such as data.gov[1], datahub[2] and data.world[3]. Most of those data portals use CKAN[4] as their backend. However, there are two problems of dataset search engines using such infrastructure: First, ranking performance relies on the quality of metadata of datasets, while many datasets lack high quality metadata; second, the information in the metadata may not satisfy the user's information need or help them solve their task [47]. A user may not know the organization of a potentially relevant dataset, or the tags data publishers provide with a dataset. Such information can hardly be used for dataset ranking.

In this chapter, we focus on the problem of dataset retrieval where dataset content is in tabular form, since tabular data is widely-used and easy to read and write. As illustrated in Fig. 5.1, a dataset consists of a data table (dataset content) and metadata. A data table usually has one header row, followed by one or more data rows. The header row consists of a list of **schema labels** (attribute names) whose actual values are stored in data rows. Metadata usually includes title and description of the dataset.

Schema labels, which represent high-level concepts, are underutilized if we directly score them with a user query. Consider the example in Fig. 5.1; the vocabulary of schema labels could be very different from other fields and user queries. "LocationAbbr", standing for "Location Abbreviation", is unlikely to appear in a user query so this dataset is less likely to be recalled. However, we can enhance this dataset by generating schema labels such as "place" and "city" appearing in other, similar datasets, which could provide a better soft-matching signal with respect to a user query, and therefore increase the chance that it can be recalled.

In the rest of this chapter, we first propose a new method for schema label generation. We learn latent feature representations of schema labels automatically by jointly decomposing the dataset-schema label interaction matrix and schema label-schema label interaction matrix. Then we propose a framework for enhancing

[1]`data.govhttps://www.data.gov/`
[2]`http://datahub.io/`
[3]`https://data.world/`
[4]`https://docs.ckan.org/`

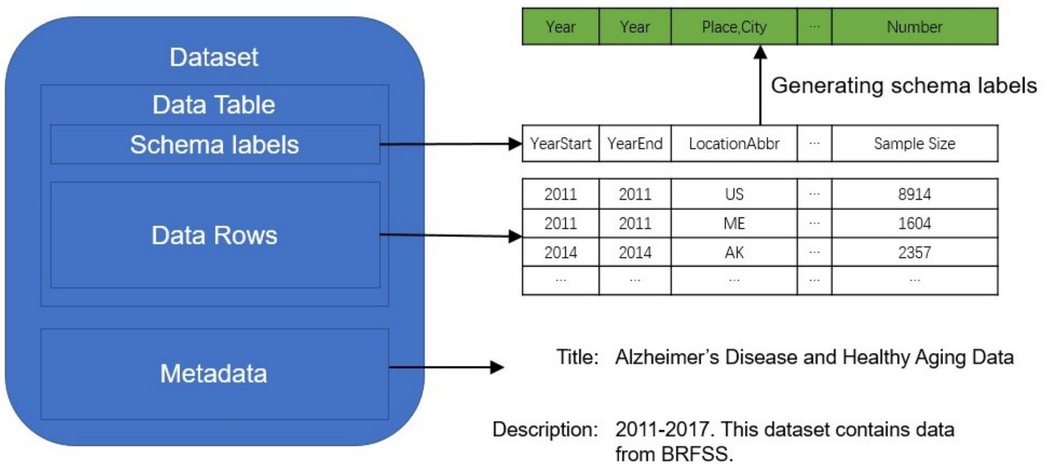

Dataset
Data Table
Schema labels
Data Rows
Metadata
Year | Year | Place,City | ··· | Number
Generating schema labels
YearStart | YearEnd | LocationAbbr | ··· | Sample Size
2011 | 2011 | US | ··· | 8914
2011 | 2011 | ME | ··· | 1604
2014 | 2014 | AK | ··· | 2357
Title: Alzheimer's Disease and Healthy Aging Data
Description: 2011-2017. This dataset contains data from BRFSS.

Figure 5.1: The structure of a dataset. Metadata includes the title and any description. A trained schema label generator is used to generate additional schema labels (green part) from similar data tables.

dataset retrieval by schema label generation to address the problem that schema labels are not effectively used by existing dataset search engines. We create a new public benchmark[5] based on federal (U.S.) datasets and use it to demonstrate the effectiveness of our proposed framework for dataset retrieval. We additionally consider a web table retrieval task and demonstrate that the features generated from schema labels can be effective for supervised ranking.

5.2 Methods

In this section, we introduce the framework of schema label enhanced dataset retrieval. As illustrated in Fig. 5.2, our framework has two stages: in the first stage, we first train a schema label generator with the method proposed in Section 5.2.1 and use it to generate additional schema labels for all the datasets; in the second stage, we use a mixed ranking model to combine the scores of schema labels and other fields for dataset ranking. In the following subsections, we present a detailed

[5]Available via `https://github.com/Zhiyu-Chen/ECIR2020-dataset-search`

illustration of the two stages.

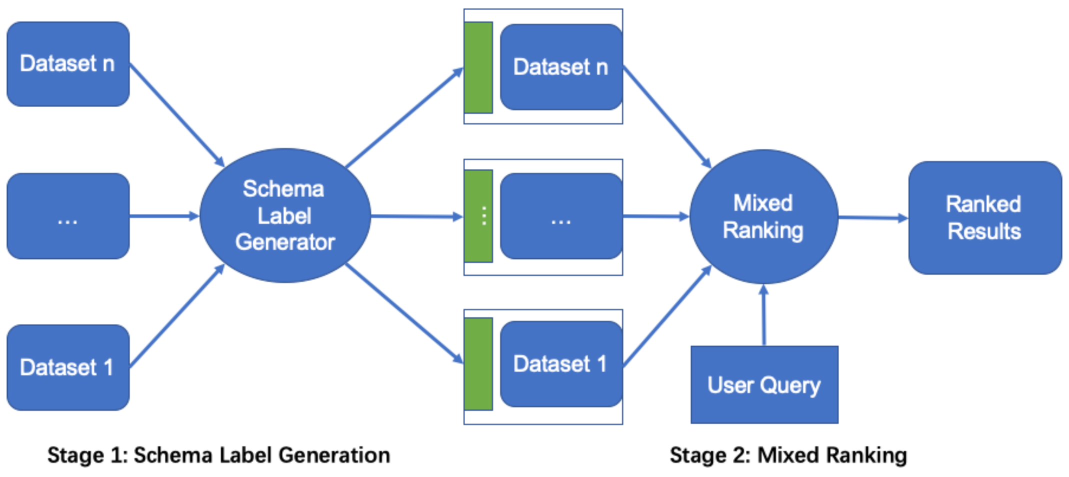


Figure 5.2: The proposed schema label enhanced dataset retrieval framework. The green blocks indicate generated schema labels for different datasets.

5.2.1 Schema Label Generation

We propose to improve dataset search by making use of generated schema labels, since these can be complementary to the original schema labels and especially valuable when they are otherwise absent from a dataset.

We treat schema label generation as a multi-label classification problem. Let $L = \{l_1, l_2, ..., l_k\}$ denote the labels appearing in all datasets and $D = \{(\mathbf{x}^i, \mathbf{y}^i) | 1 \leq i \leq n\}$ denote the training set. Here, for each training sample $(\mathbf{x}^i, \mathbf{y}^i)$, $\mathbf{x}^i$ is a d-dimensional feature vector of column i which can be calculated from data rows [54] or learned from matrix factorization proposed later in this section. $\mathbf{y}^i$ is k-dimensional vector $[y_1^i, y_2^i, ..., y_k^i]$ and $y_j^i = 1$ only if x_i is relevant to label l_j, otherwise $y_j^i = 0$. Our objective is to learn a function that models $P(l|x_i)$, $(l \in L)$. To generate m schema labels for column i, we can select the top m labels L_m by:

$$L_m = \underset{l \in L_m \subseteq L}{\arg\max}\, P(l|x_i)$$

We could also generate schema labels by selecting a probability threshold θ:

$$L_m = \{l \in L | P(l|x_i) \geq \theta\}$$

In practice, we could first generate the top m schema labels and filter out those results with a probability lower than the threshold.

In Chapter 4, we proposed to predict schema labels based on curated features of data values. Instead of designing curated features for schema labels, we consider learning their representations in an automated manner. Inspired by collaborative filtering methods in recommender systems, we model each dataset as a user and each schema label as an item. Then a dataset with a schema label can be considered as positive feedback between a user and an item. By exploiting the user-item co-occurrences and item-item co-occurrences, we can learn the latent representations of schema labels. In the following, we show how to construct a preference matrix in the context of schema label generation and how to learn the schema label features.

Preference Matrix Construction. With m data tables and n unique schema labels, we can construct a dataset-column preference matrix $M^{m \times n}$, where M_{up} is 1 if dataset u contains schema label p.

Matrix Factorization. MF [126] decomposes M into the product of $U^{m \times k}$ and $P^{k \times n}$ where $k < min(m, n)$. U^T can be denoted as $(\alpha_1, ..., \alpha_u ..., \alpha_m)$ where $\alpha_u \in R^k$ represents the latent factor vector of dataset u. Similarly, P^T can be denoted as $(\beta_1, ..., \beta_p ..., \beta_n)$ where $\beta_p \in R^k$ represents the latent factor vector of schema label p. Since the preference matrix actually models the implicit feedback, MF optimizes the following objective function:

$$\mathcal{L}_{mf} = \sum_{u,p} c_{up}(M_{up} - \alpha_u^T \beta_p)^2 + \lambda_\alpha \sum_u \|\alpha_u\|^2 + \lambda_\beta \sum_p \|\beta_p\|^2 \tag{5.1}$$

where c_{up} is a hyperparameter tuned to balance the non-zero and zero values since M is a sparse matrix. λ_α and λ_β are regularization parameters that adjust the importance of regularization terms $\sum_u \|\alpha_u\|^2$ and $\sum_p \|\beta_p\|^2$.

Label Embedding. Recently, word embedding techniques (e.g., word2vec

[167]) have been valuable in natural language processing tasks. Given a sequence of words, a low-dimensional continuous representation called word embedding can be learned for each word. Word2vec's skip-gram model with negative sampling (SGNS) is equivalent to implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant [142]. The PMI between word i and its context word j is defined as:

$$PMI(i,j) = log\frac{P(i,j)}{P(i) \times P(j)} = log\frac{\#(i,j) \times |D|}{\sum_j \#(i,j) \times \sum_i \#(i,j)}$$

where $\#(i,j)$ is the number of times word j appears in the context window of word i and $|D|$ is the total number of word-context pairs. Then, a shifted positive PMI (SPPMI) of word i and word j is calculated as:

$$SSPMI(i,j) = max\{PMI(i,j) - log(k), 0\} \tag{5.2}$$

where k is the number of negative samples of SGNS. Given a corpus, matrix M^{SPPMI} can be constructed based on equation (5.2) and factorizing it is equivalent to performing SGNS.

A schema label exists in the context of other schema labels. Therefore, we perform word embedding techniques to learn the latent representations of schema labels. However, we do not consider the order of schema labels. Therefore, given a schema label, all other schema labels which come from the same data table are considered as its context. With the constructed SSPMI matrix of co-occurring schema labels, we are able to decompose it to learn the latent representations of schema labels.

Joint Learning of Schema Label Representations. Schema label representations learned from MF capture the interactive information between datasets and schema labels, while the word2vec style representations explain the co-occurrence relationships of schema labels. We use the CoFactor model [146] to jointly learn schema label representations from both dataset-label interaction and label-label interaction:

$$
\begin{aligned}
\mathcal{L} = & \overbrace{\sum_{u,p} c_{up}(M_{up} - \alpha_u^T \beta_p)^2}^{MF} \\
& + \overbrace{\sum_{M_{pi}^{SPPMI} \neq 0} (M_{pi}^{SPPMI} - \beta_p^T \gamma_i - b_p - c_i)^2}^{schema\ label\ embedding} \\
& + \lambda_\alpha \sum_u \|\alpha_u\|^2 + \lambda_\beta \sum_p \|\beta_p\|^2 + \lambda_\gamma \sum_i \|\gamma_i\|^2
\end{aligned}
\tag{5.3}
$$

From the objective function we can see the schema label representation β_p is shared between MF and schema label embedding. γ_i is the latent representation of context embedding. b_p and c_i are the schema label embedding bias and context embedding bias, respectively. The last line of Equation 5.3 incorporates regularization terms with different λ controlling their effects. We use the vector-wise ALS algorithm [289] to optimize the parameters.

Schema label generation. After obtaining the jointly learned representations of schema labels, we can use them as features for schema label generation. In this chapter, we use the concatenation of schema label representations introduced here and the curated features proposed by Chen et al. [54] to construct each x^i. Any multi-label classification models can be used to train the schema label generator and in this chapter we choose Random Forest.

5.2.2 The Mixture Ranking Model

Based on the schema label generation method proposed above, we index the generated schema labels for each dataset. Now, each dataset has the following fields: metadata, data rows, schema labels and generated schema labels. A straightforward way to rank datasets is to use traditional ranking methods for documents.

Zhang and Balog [299] represent tables as single field documents or multifield documents for table retrieval task. For *single field document representation*, a dataset is treated as a single document by concatenating the text from all the fields. Then traditional methods such as BM25 can be used to to score the dataset. For *multifield document representation*, each field is scored independently against the query and a weighted sum is used for ranking.

In our **Schema Label Mixed Ranking (SLMR)** model, we score schema labels differently from other fields. The focus of our work is to learn how schema labels, data rows and other metadata may differently influence dataset retrieval performance. Note that, for simplicity, we consider the other metadata (title and description) as a single text field, since title and description are homogeneous compared with schema labels and data rows. Therefore, we have the following scoring function for a dataset D:

$$score(q, D) = \sum_{i \in \{text, data\}} w_i \times score_{text}(q, F_i) + w_l \times score_l(q, F_l) \tag{5.4}$$

where F_{text} denotes the concatenation of title and description, F_{data} denotes the data table, and F_l denotes the generated schema labels. Each field has a corresponding weights. F_{text} and F_{data} have the same scoring function $score_{text}$ while F_l has a different scoring function $score_l$. For F_{text} and F_{data}, we can use a standard scoring function for normal documents. In the experiments, we use BM25 as $score_{text}$.

Due to the existence of a large number of non-dictionary words in schema labels [54] that would otherwise be outside of the vocabulary of a word-based embedding, we represent schema labels and query terms using fastText [28] in $score_l$, since such word embeddings are calculated from character n-grams instead of terms. To score the schema labels with respect to a query, we use the negative Word Mover's Distance (WMD) [131]. WMD measures the dissimilarity between two text documents as the minimum amount of distance that the word embeddings of one document need to "travel" to reach the word embeddings of another document. So $score_l(q, F_l) = -wmd(fasttext(q), fasttext(F_l))$ reflects the semantic similarity between a query and schema labels.

Table 5.1: NDCG@k and Precision@k of different models on dataset retrieval. The superscript + shows statistically significant improvements for our SLMR model over other single and multifield document ranking models. T means title, D means description, DT means data table, G means generated schema labels.

Method	Used Fields	NDCG@5	@10	@20	@50	P@5	@10	@20	@50
SDR	T+D	0.8920	0.8490	0.8222	0.8121	0.4122	0.3652	0.3452	0.3585
SDR	DT	0.7378	0.7036	0.6964	0.7107	0.2856	0.2974	0.2931	0.3122
SDR	T+D+DT	0.8435	0.7954	0.7763	0.7785	0.2574	0.2870	0.3170	0.3357
MDR	T+D+DT	0.9285	0.8874	0.8683	0.8631	0.4086	0.3612	**0.4026**	0.3767
SLMR	T+D+G	**0.9293**$^+$	**0.8898**	**0.8722**$^+$	**0.8662**	**0.5000**$^+$	**0.4388**$^+$	0.4000	0.3761
SLMR	T+D+DT+G	0.9169	0.8808	0.8680	0.8555	**0.5000**$^+$	0.4345$^+$	0.4013	**0.3783**

5.3 Experiments

5.3.1 Evaluation Metrics

We evaluate dataset retrieval performance over a range of metrics: Precision at k and Normalized Discounted Cumulative Gain (NDCG) at k [113]. To test the significance of differences between model performances, we use paired t-tests with significance at the $p = 0.01$ level.

5.3.2 Baselines

We first present the baseline retrieval methods.

Single-field document ranking (SDR). A dataset is considered as a single document. We use BM25 to score the concatenation of title and description, the text of the data table and the concatenation of all of them. By comparing the three results, we can learn about field level importance for dataset retrieval. Parameters are chosen by grid search.

Multifield document ranking (MDR). By setting $w_l = 0$, Eq. (5.4) degenerates to the Mixture of Language Models [187]. BM25 is also used here as $score_{text}()$ in order to have a fair comparison with other methods. To optimize field weights, we use coordinate ascent. Finally, smoothing parameters are optimized in the same manner as single-field document ranking.

5.3.3 Experimental Results

In this section, we examine the following research questions:

Q1 Does data table content help in dataset retrieval?

Q2 Do generated schema labels help in dataset retrieval?

Q3 Which fields are most important for the dataset retrieval task?

We first obtain features of schema labels as described in Section 5.2.1 and the number of latent factors is set to 40. Then we train a Random Forest with the learned schema label features. The scikit-learn implementation of Random Forest[6] is used with default parameters except the number of trees is set to 25. In practice, we could choose any multi-label classifier. For each column, we select the top 10 generated schema labels and filter those with probability lower than 0.5. For each dataset, we index the generated schema labels as an additional field. Table 5.1 summarizes the NDCG at k and Precision at k of different models. Note that, for Schema Label Mixed Ranking (SLMR), we trained three different models and the weights of used fields were forced to be non-zero in order to study the proposed research questions. The weights of used fields for multifield document representation are also set non-zero when optimizing the parameters.

From the results of single-field document ranking, we can see that only utilizing the data table for ranking leads to the worst performance. Scoring on the concatenation of title and description achieved the best results, which indicates that title and description are more important than the data table for ranking a dataset (**Q3**). Treating all fields of a dataset as a single-field document provides performance between the previous two models. This result is expected since the length of data tables are usually much larger than titles and descriptions, and therefore dominate the table representation.

By comparing the results of single-field and multifield document ranking, we observe that the combination of the scores of data table, title and description could

[6]`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifer.html`

improve NDCG@k. Though NDCG@k decreases when k increases, the relative improvement against single-field document ranking are more significant. In contrast, for Precision@5, Precision@10, single-field document ranking performs better than multifield document ranking, though the differences are small. So for **Q1**, under the setting of multifield document ranking, the content of the data table could help NDCG, but not help Precision of dataset retrieval results.

Without scoring data tables, our proposed schema label mixed ranking approach achieves the highest NDCG on all the rank cut-offs, which indicates that the generated schema labels can be useful to improve the NDCG of dataset retrieval results (**Q2**). Though Precision@20 of multifield document ranking are higher than our proposed model, the difference is no more than 0.4% ($p\_value > 0.9$). Significantly, our model outperforms by 21.3% for Precision@5 ($\frac{0.5-0.4122}{0.4122}$) and by 20.1% for Precision@10 ($\frac{0.4388-0.3652}{0.3652}$) than the best baseline methods ($p\_value < 0.01$). Whether data tables are scored or not, Precision@k is not significantly different for schema label mixed ranking. Therefore, under the setting of schema label mixed ranking, data tables make little contribution in this scenario (**Q1**). One possible reason could be that data tables collected from data.gov contain large quantities of numerical values and will rarely be used to match user queries.

If a schema label mixed ranking model scores only on titles and descriptions ($w_l = 0$), it is equivalent to single-field ranking model scoring on titles and descriptions. Therefore, we can compare the results in first and fifth rows in Table 5.1. With generated schema labels, the ranking model can have a higher performance on dataset retrieval task (**Q2**).

5.3.4 Schema Label Generation Enhanced Search for Web Tables

The task of dataset search is similar to Web table search since both tasks use table structure to represent data. The difference is that a large amount of Web tables are entity focused and contain many named entities that can be linked to a knowledge base. However, our datasets collected from the data.gov data portal contain few

Table 5.2: Supervised ranking results on table retrieval.

Method	NDCG@5	@10	@15	@20
STR[298]	0.6366	0.6571	0.663	0.6632
Schema Label Features	0.4489	0.5201	0.534	0.5347
STR + Schema Label Feat.	**0.6530**	**0.6728**	**0.6789**	**0.6761**

useful entities in the table. Therefore, a lot of methods designed for Web table ranking cannot be applied to dataset search. The semantic table retrieval (STR) method proposed by Zhang and Balog [298] relies on features from knowledge bases (bag of entities) which are not generally available for the scenario of dataset search. However, the schema label generation based method can be applied to table search. Thus, we performed additional experiments to show the performance of our method for the table search scenario.

We first generate schema labels for the table corpus shared by Zhang and Balog [298] using the method proposed in Section 5.2.1. Then we append five additional features to their proposed features[7] based on schema labels. Each feature is one type of semantic similarity between query and schema labels. Four features are calculated using the measurement proposed by Zhang and Balog (one early fusion feature, three late fusion features) and the last feature is the negative of Word Mover's Distance. Finally, like Zhang and Balog, we use Random Forest to perform pointwise regression and the final reported results are averaged over five runs of 5-fold cross-validation and shown in Table 5.2.

We can see that schema label features along cannot outperform STR. But combining them results in improvement. However, by calculating the normalized feature importance measured in terms of Gini score, we find that for STR with schema label features, WMD based measurement contributes the most among all the semantic features. Thus it demonstrates that the schema labels can be valuable for the table retrieval task as well.

[7] `https://github.com/iai-group/www2018-table/tree/master/feature`

Table 5.3: Unsupervised ranking results on table retrieval.

Used Fields	NDCG@5	@10	@15	@20
text	0.3724	0.3891	0.4009	0.4178
text + data table	0.3901	0.4042	0.4422	0.4686
text + data table + generated labels	**0.4006**	**0.4118**	**0.4495**	**0.4766**
text + data table + original labels	0.3930	0.4055	0.4457	0.4709
text + original labels	0.3785	0.3934	0.4110	0.4283
text + generated labels	0.3808	0.3955	0.4064	0.4197

Notably, in this table corpus, many tables lack much table content but contain rich text descriptions, which could be unfair for schema label generation-based methods. While for dataset search, each table has values but may lack high quality dataset descriptions. We believe that our schema label generation method can outperform STR in the scenario where text descriptions provide less useful information than the table itself.

We also show unsupervised ranking results with Equation 5.4 in Table 5.3. Unlike Zhang and Balog [298], we consider page title, section title and caption as a single text field, in order to reduce the number of hyperparameters (field weights). The results show that generated labels are more effective than original labels for table ranking. It is unsurprising because generated labels often include not only original labels but also additional labels that can benefit the ranking model. We also notice that including the data table field achieves better results than not scoring it, which is contrary to the results of dataset ranking. It is also expected since WikiTables are entity-focused and include a lot of text information while data tables from data.gov include more numeric values.

5.4 Summary

In this chapter, we have proposed a schema label enhanced ranking framework for dataset retrieval. The framework has two stages: in the first stage, a schema label generator is trained to generate additional schema labels for each dataset column;

in the second stage, given a user query, datasets are ranked by their original fields together with generated schema labels. Schema label generation is treated as a multi-label classification task in which each column of a dataset is associated with multiple schema labels. Instead of using hand-curated features, we learn the latent feature representations of schema labels by a CoFactor model in which the dataset-schema label interactions and schema label-schema label interactions are captured. With the schema label mixed ranking model, the traditional ranking scores for text fields (title, description, data rows) and word embedding-based scores for generated schema labels can be used to rank the datasets.

We created a new benchmark to evaluate the performance of dataset retrieval. The experimental results demonstrate our proposed framework can effectively improve the performance on the dataset retrieval task. It achieved the highest NDCG on all the rank cut-offs compared with all baseline methods. We also apply our method to the web table retrieval task which is similar to dataset search and find that the features generated from schema labels can help in supervised ranking as well.

5.5 Bibliographic Notes

Related work in this chapter is in two topics: retrieval of multifield documents and recommendation systems.

Information retrieval for structured documents has been studied in the past. Considering the structure of a document when designing retrieval models can usually improve retrieval results. It has been shown that combining similarities and rankings of different sections can lead to better performance [275]. Ogilvie et al. [187] presents a mixture-based language model combining different document representations for known-item search in structured document collections.They find that document representations that perform poorly can be combined with other representations to improve the overall performance. Robertson et al. [223] introduces BM25F which is an extension of BM25 and combines original term frequencies in the different fields in a weighted manner. A field relevance model is proposed by Kim and Croft [121] to incorporate relevance feedback for field weights estimation. There are also supervised methods for multifield document ranking. A Bayesian networks-based model for structured documents was proposed by Piwowarski and Gallinari [202]. Kim et al. [122] proposed a probabilistic model for semi-structured document retrieval. They calculate the mapping probability of each query term and use it as a weight to combine the language models estimated from each field. Svore et al. [244] developed LambdaBM25, a machine learning approach to BM25-style retrieval that learns from the input attributes of BM25 and performs better than BM25F for multifield document ranking. Zamani et al. [291] proposed a neural ranking model that learns an aggregated document representation from field-level representations and then uses a matching network to produce the final relevance score. In this chapter, a dataset consists of multiple components (as described in Figure 5.1), and therefore dataset search can be treated as a multifield document retrieval task.

Recommendation systems have been widely used in e-commerce applications. There are three types of recommendation algorithms: content-based filtering, collaborative filtering and hybrid filtering. For content-based filtering [17, 196], a recommendation is made based on user profiles and item descriptions. An item is recommended if it is mostly related to the positively rated items. To build collaborative filtering recommendation system, data about user activities is required in order to construct the user-item preference matrix. Based on the preference matrix, it learns to recommend items from the most similar users. Matrix Factorization (MF) is widely used in collaborative filtering. Basic MF models (e.g., [126, 169]) learn the latent features of users and items. The dot product result of a user latent feature and an item latent feature models the rating score for a given user-item pair. Hybrid filtering [98, 189] combines content-based filtering and collaborative filtering which could help overcome cold start and the sparsity problem. In this chapter, we borrow from recommendation methods. By modeling the dataset-label interaction and label-label interaction, we learn the latent features for each schema label.

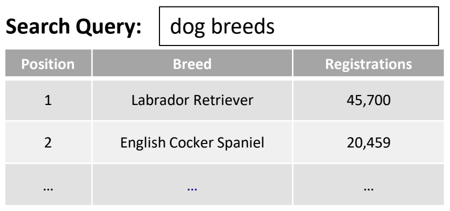
Chapter 6

Dataset Search with Pretrained Language Models

In this chapter, we use the deep contextualized language model BERT for the task of ad hoc table retrieval. Pretrained contextualized language models such as BERT have achieved impressive results on various natural language processing benchmarks. Benefiting from multiple pretraining tasks and large scale training corpora, pretrained models can capture complex syntactic word relations. We investigate how to encode table content considering the table structure and input length limit of BERT. We also propose an approach that incorporates features from prior literature on table retrieval and jointly trains them with BERT. In experiments on public datasets, we show that our best approach can outperform the previous state-of-the-art method and BERT baselines with a large margin under different evaluation metrics.

6.1 Introduction

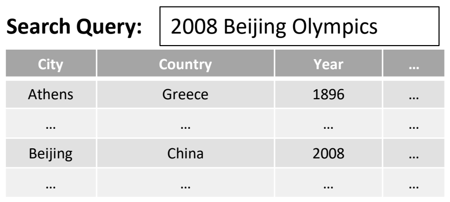
As an efficient way to organize and display data, tables are broadly used in different applications: researchers use tables to present their experimental results; companies store information about customers and products in spreadsheets; flight information display systems in the airports show flight schedules to passengers in tables. According to Cafarella et al. [36], there are more than 14.1 billion tables on the Web. Among those tables, many are very informative which means they include relations and attributes of real-world entities, and have been used for a variety of downstream tasks. For example, tables like Wikipedia infoboxes have been used to construct knowledge bases since they are of high quality and consistent structure [13]. Data-to-text models take tables from specific domains as input and transform them into fluent natural language sentences such as sports news [276] and product descriptions [44]. With structure information and metadata, tables store factual knowledge and therefore are also used to build question answering (QA) systems [241].



Search Query: dog breeds

Position	Breed	Registrations
1	Labrador Retriever	45,700
2	English Cocker Spaniel	20,459
…	…	…

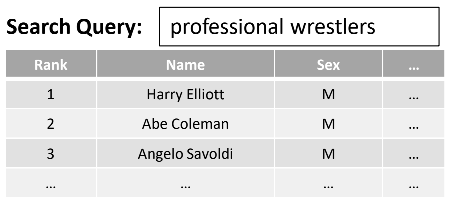
(a) An example of a returned table in which one column is relevant to the query.



Search Query: 2008 Beijing Olympics

City	Country	Year	…
Athens	Greece	1896	…
…	…	…	…
Beijing	China	2008	…
…	…	…	…

(b) An example of a returned table in which one row is relevant to the query.



Search Query: professional wrestlers

Rank	Name	Sex	…
1	Harry Elliott	M	…
2	Abe Coleman	M	…
3	Angelo Savoldi	M	…
…	…	…	…

(c) An example of a returned table in which all cells are relevant to the query.

Figure 6.1: Three examples of returned tables reflecting different relevant unit types.

The table retrieval task is related but different from the table QA task. Both of them aim to satisfy users' information need. QA models usually take a natural language question as input and aim to find one or more specific answers. However, queries for table retrieval systems may have ambiguous intent and usually consist of several keywords. The returned tables from a table retrieval system in Figure 6.1a and 6.1c can be both positive samples for a table QA system. A user may want to know the list of all dog breeds in Figure 6.1a and the 2nd column of the table provides the relevant and accurate information. A user may ask the profiles of professional wrestlers in Figure 6.1c and the returned table contains that information. For this example, all the cells provide informative content for the user. The 2nd column tells who are professional wrestlers and the other columns provide context information. However, in Figure 6.1b, the query has more ambiguous intent. The user may ask the results of 2008 Beijing Olympic Games which means the returned table is a negative sample for a QA system. If a user does not have a clear question and just wants to explore what he/she could find, then the returned table is a positive sample for a table retrieval system. For this example, the row that includes "Beijing" is relevant and the remaining rows are less useful. We note that the unit of relevant information in the table can be rows, columns or cells. Based on this observation, we propose different methods to select row items, column items and cell items from a table.

In this chapter, we consider the task of ad hoc table retrieval where given a keyword query, a list of ranked tables are returned. In previous studies of table retrieval, various features are used. Word level, phrase level and sentence level features are calculated by Sun et al. [242]. Zhang et al. [299] use 23 hand-crafted features and 16 embedding based features to train a random forest for pointwise table ranking. Recently, the pre-trained language model BERT [77] and its variants like RoBERTa [153] have achieved impressive results on different natural language understanding tasks [261]. The self-attention structure and pre-training tasks enable BERT to learn complex linguistic features from a large corpus. Researchers from IR communities have applied BERT to ranking tasks and achieved new state-of-the-art results on multiple benchmarks [284, 285, 159, 185]. Here we apply BERT to the ad hoc table retrieval task. In previous work, the input of BERT is either a single sequence or sequence pairs. The question of how to effectively encode a structured document into a BERT representation has not been previously explored. We construct input for BERT considering the structure of a table and then combine BERT features with other table features together to treat table retrieval as a regression task.

We summarize our contributions as the following:

- We propose three content selectors to encode different table items into the fixed-width BERT representation.
- We experiment on multiple datasets and demonstrate that our method achieves the best results and generalizes to other domains.
- We analyze the experiment results and discuss why the max salience selector for row items performs the best among all other methods.
- We analyze the fine-tuned BERT attention maps and embeddings, and explain what information is captured by BERT.
- We propose a new test collection for Web table retrieval which has not only relevance judgments of query-table pairs, but also the relevance judgments of query-table context pairs with respect to a query.

6.2 Prerequisites

6.2.1 BERT

BERT [77], consisting of L layers of Transformer blocks, is a deep contextual language model which has achieved impressive results on various natural language processing tasks. Given a sequence of input token embeddings $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, the Transformer block at layer l outputs the contextualized embeddings (hidden states) of input tokens $\boldsymbol{H}^l = \{\boldsymbol{h}_1^l, \boldsymbol{h}_2^l, ..., \boldsymbol{h}_n^l\}$. The Transformer block is originally proposed by Vaswani et al. [256] and each has the same structure: multi-head self-attention followed by a feed-forward network.

$$\begin{aligned} & Transformer_l(\boldsymbol{H}^{l-1}) \\ & = FFN(MH\_Attn(\boldsymbol{H}^{l-1})) \\ & = FFN(\boldsymbol{W}[Attn_1(\boldsymbol{H}^{l-1}), ..., Attn_m(\boldsymbol{H}^{l-1})]) \end{aligned} \tag{6.1}$$

Multi-head self-attention aggregates the output from m attention heads.

When using BERT for downstream tasks, special tokens ([SEP] and [CLS]) are added into the input. For single sequence classification/regression tasks, [CLS] and [SEP] are added to the beginning and end of the input sequence. For sequence-pair classification/regression, the two input sequences are concatenated by [SEP] and then processed the same as single sequence tasks. The embedding of [CLS] from the last Transformer block is fed into a final classification/regression layer.

6.2.2 BERT Characteristics

Limit on input length. BERT cannot take input sequences longer than 512 tokens. In previous studies of BERT for long document tasks like text classification [46], the input tokens are truncated. Better ways to preprocess the inputs beyond length limitation are worth studying since trivially throwing away part of the inputs could lose important information. Transformer-XL [70] solves the fixed-length issue with recursion and relative position encoding. However, this method requires further pre-training and is only evaluated on text generation tasks. Though we focus on table retrieval, our methods to alleviate the long sequence issue are off-the-shelf without any further training and can also be applied to other domains.

The secrets behind special tokens. Before BERT was proposed, neural models for NLP and IR tasks usually take the embeddings of all input tokens for training. While for BERT and its variants, fine-tuning on the target tasks only requires an additional softmax layer on top of the [CLS] embedding from the last layer and the remaining embeddings are not used. The function of [SEP] is often disregarded, as when constructing the input of BERT, the role of [SEP] is just a symbol to mark the end or delimiter of a sequence. Recently, researchers begin to analyze why BERT is so effective for different tasks. Clark et al. [61] suggest that [SEP] might be used as a "no-op"sometimes and does not aggregate segment-level information. However, Ma et al. [157] show that using the embedding of [SEP] instead of [CLS] can also achieve comparable results, which indicates that [SEP] also learns contextualized information of the sequence. In our experiments, we study the relationship between special tokens and other input tokens in order to explore what BERT embeddings learn after fine-tuning on the target task.

6.3 Methods

Here we define the task and then describe our method in detail.

6.3.1 Task Definition

In ad hoc table retrieval, given a query $q \in Q$ usually consisting of several keywords $q = \{k_1, k_2, ..., k_l\}$, our goal is to rank a set of tables $T = \{t_1, t_2, ..., t_n\}$ in descending order of their relevance scores with respect to q. A table is a set of cells arranged in rows and columns like a matrix. Each cell could a be single word, a real number, a phrase or even sentences. The first row of a table is the header row and consists of header cells. In practice, tables from the Web could have more complex structures [65]. In this chapter, we only consider tables that have the simplest structure since they are the most commonly used. Each table could have context fields $\{p_1, ..., p_k\}$ depending on the source of the table. For example, a table from Wikipedia can have a caption, its section title and page title.

6.3.2 BERT for Table Retrieval

We show the overview of our framework which includes four components in Figure 6.2. The content selector extracts informative items (rows, columns or cells) from a table. BERT is used to extract features f_{bert} from the query, corresponding table context fields, and selected items. A neural network is used to transform

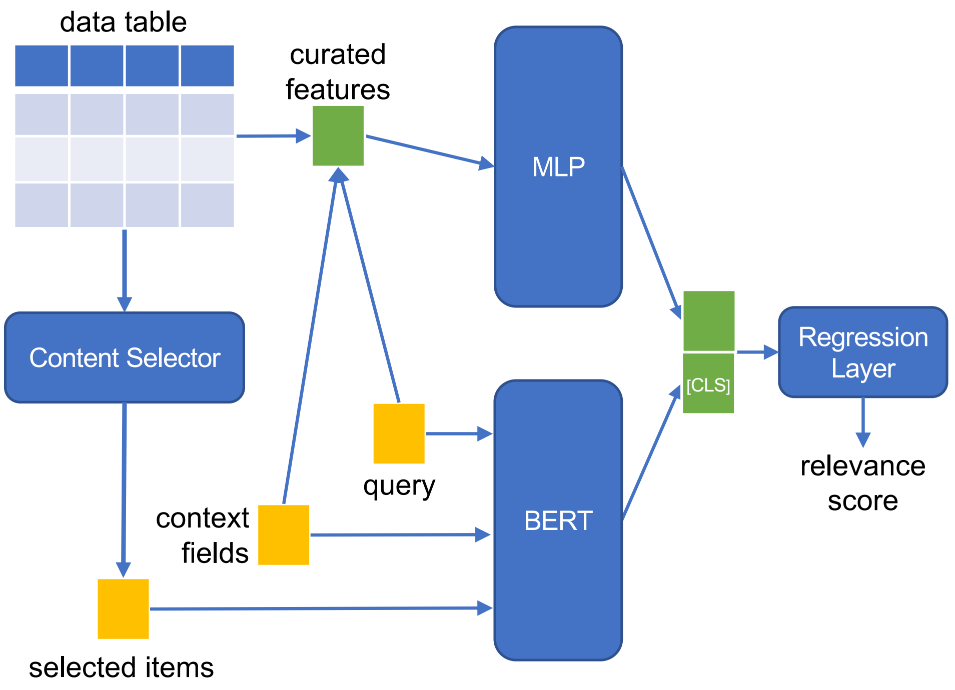


Figure 6.2: Overview of the proposed model. Blue blocks are model components. Orange blocks are raw text of the input. Green blocks are either manually curated features or outputs from models (BERT and MLP).

additional features v_a (if provided) to f_a. Then f_{bert} and f_a are concatenated into a single feature vector. This vector is fed into a regression layer to predict the relevance score. In the rest of this section, we describe the model components in detail.

Content Selector

As previously mentioned, BERT can only take input sequences that are no longer than 512 tokens. But for many tasks including table retrieval, the lengths of inputs can easily exceed that limit. To deal with the limit for various downstream tasks, inputs are typically truncated into valid lengths or multiple instances for a single document are created [132, 284]. In open-domain question answering and machine reading comprehension, a single instance usually involves long documents or multiple documents, and the proposed methods usually have a select-then-extract two-stage schema [272, 106, 149, 136, 274]. Inspired by those works, we propose a **select-then-rank** framework for ad hoc table retrieval. First, we select a set of potentially informative items from a table. Then we pack the context fields of a table and its selected items as the final table representation. Finally we extract BERT features f_{bert} for all the tables based on the new constructed representation.

In the ad hoc table retrieval task, we notice that there are three types of relevant tables in terms of the unit of the relevant information:

- One or more columns are relevant to the query. For example, only the second column is relevant to the query in Figure 6.1a;
- One or more rows are relevant to the query. For example, only the row that includes "Beijing" is relevant to the query in Figure 6.1b.
- The relevant information is spread over the whole table. For example, in Figure 6.1c, the table includes a list of records about the entities asked by the query.

Therefore, we slice a table t into a list of items $\{c_1, ..., c_m\}$, i.e., a list of rows, columns or cells and select the top-ranked items for the final BERT input representation. Here we propose three methods to measure the salience score of a table item c:

- Mean Salience: it assumes that the relevance signal can be captured by the similarity of query representation and item representation. We use the average word embeddings to represent queries and items respectively.

$$SAL_{mean}(c) = cosine(\frac{\sum_{w \in c} v_w}{l_c}, \frac{\sum_{k \in q} v_k}{l_q})$$

- Sum Salience: it assumes relevance signals between every pair of query and item terms are useful for content selection.

$$SAL_{sum}(c) = \sum_{k \in q} \sum_{w \in c} cosine(v_k, v_w)$$

- Max Salience: it assumes that only the most salient signal between any pair of query and item terms is useful for content selection.

$$SAL_{max}(c) = \max_{k \in q, w \in c} cosine(v_k, v_w)$$

Instead of trivially truncating table information, we rank the items of a table and keep items with higher salience scores in the front.

Encoding Table for BERT

Given a query $q \in Q$, a table $t \in T$, the context fields $\{p_1, ..., p_k\}$ and selected items of t $\{c_1, ..., c_m\}$, we construct the final input sequence for BERT as

$$S = [[CLS], q, [SEP], p_1, [SEP], ..., p_k, [SEP], c_1, [SEP], ..., c_m, [SEP]]$$

Like Hu et al. [77], we use WordPiece tokenization for input sequences and the input representation of each token is constructed by summing its token embedding, segment embedding and position embedding. All the queries share the same segment embedding and context fields, selected items share another segment embedding. As illustrated in Section 6.2.1, we use the embedding of $[CLS]$ from the last layer as BERT features f_{bert}.

Feature Fusion and Prediction

Feature-based methods have shown impressive performance and achieved previous state-of-the-art results on ad hoc table retrieval [299]. When additional feature $v_a \in \mathbb{R}^d$ for a query-table pair is available, we combine them with BERT features

f_{bert} by:

$$f_a = v_a \boldsymbol{W}_1 + b_1 \tag{6.2}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d}$. Then f_a and f_{bert} are concatenated into single vector and fed to the final regression layer.

$$f = [f_a; f_{bert}] \tag{6.3}$$

$$score = Regression(f) \tag{6.4}$$

When only BERT features are available, f equals f_{bert}. A simple linear transformation is used as regression layer which means $Regression(f) = f\boldsymbol{W}_2 + b_2$ where $\boldsymbol{W}_2 \in \mathbb{R}^{(d+h) \times 1}$ and h is the size of BERT hidden states.

Training

We use the pre-trained BERT-large-cased model which consists of 24 layers of Transformer blocks, 16 self-attention heads per layer and has a hidden size of 1024. Considering processing speed, the size of GPU memory, and the fact that BERT is good for short text tasks, the maximum input length is set to 128. Since the selected items are at the end of the input (as described in Section 6.3.2) and ranked by their salience scores with respect to the query, we assume the truncated part will have the least negative impact with a given length constraint. Considering the dataset statistics in Table 6.1, we limit the caption to 20 tokens, section title and page title to 10 tokens each, and table headers to 20 tokens. Since queries are short, we keep all the query tokens. As a result, we leave about half of the space for table content. We fine-tune the framework by minimizing the Mean Square Error (MSE) between model predictions and gold standard relevance scores.[1] We train the model with 5 epochs and batch size of 16. The Adam optimizer with learning rate of 1e-5 is used. We also use a linear learning rate decay schedule with warm-up of 0.1. Our implementation is based on code from an open source repository.[2]

[1]We also tried binary classification to predict relevance probabilities as in Sakata et al. [226] and found that regression is much better in our scenario.

[2]`https://github.com/huggingface/transformers`

Table 6.1: The length statistics of data provided by Zhang and Balog [299]. The length is calculated after WordPiece tokenization.

Field	Mean	Max	> 512	> 128
query	3.5	8	-	-
caption	4.3	76	-	-
page title	5.6	26	-	-
section title	3.3	22	-	-
header	19.7	729	0.032%	2%
table	549.1	20545	24.2%	65.3%
all	585.5	20605	27.3%	72.4%

6.4 Experiments

In this section, we aim to answer the following research questions:

RQ1: What is the performance gain of BERT with content selection methods, with respect to state-of-the-art performance?

RQ2: Could BERT with content selection methods outperform state-of-the-art performance without additional features?

RQ3: Which content selection method/item type is the most effective?

6.4.1 Dataset Description

We use the WikiTables dataset created by Zhang and Balog [299] where the previous state-of-the-art method is proposed. The table corpus is originally extracted from Wikipedia [27]. The context fields include page title and section title. From Figure 6.1b and Figure 6.1a we can see that the first row of a table usually contains some high-level concepts and provides informative context. Therefore we also consider the table header as a context field. When slicing the tables, we still have table headers included. The queries are sampled from the collections in [35, 259]. In total, they annotated 3120 query-table pairs. The statistics of the corpus are shown in Table 6.1. We also use the curated features proposed by Zhang and Balog [299] for feature fusion.

6.4.2 Experimental Setup

The performance of table retrieval methods is evaluated with Mean Average Precision (MAP), Mean Reciprocal Rank (MRR) and 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 use †/‡ to denote significance levels at p =0.05, 0.005 respectively.

Based on Section 6.3.2, we have three strategies to calculate salience scores of items and three ways to construct items (as a list of columns, rows, or cells) from a table. We list all the methods settings in Table 6.2. To obtain the salience scores, we use fastText word embeddings [28].[3] Note that a different tokenization approach is used because fastText is not pre-trained on WordPiece tokenized corpus. We replace all non-numerical and non-alphabet characters with space and simply split sequences by space. Following the same experimental setup in Zhang et al. [299], five-fold cross-validation is used when evaluating different methods. We release our code on GitHub.[4]

Table 6.2: The settings of all proposed methods, which use different item types and content selectors.

Method Name	Item type	Content Selector
Hybrid-BERT-Row-Sum	Row	Sum Salience
Hybrid-BERT-Row-Mean	Row	Mean Salience
Hybrid-BERT-Row-Max	Row	Max Salience
Hybrid-BERT-Col-Sum	Column	Sum Salience
Hybrid-BERT-Col-Mean	Column	Mean Salience
Hybrid-BERT-Col-Max	Column	Max Salience
Hybrid-BERT-Cell-Sum	Cell	Sum Salience
Hybrid-BERT-Cell-Mean	Cell	Mean Salience
Hybrid-BERT-Cell-Max	Cell	Max Salience

[3]`https://github.com/facebookresearch/fastText/`

[4]`https://github.com/Zhiyu-Chen/SIGIR2020-BERT-Table-Search`

Table 6.3: The superscript † shows statistically significant improvements for the method compared with all other methods.

Method Name	MAP	MRR	NDCG@5	NDCG@10	NDCG@15	NDCG@20
STR	0.5711	0.6062	0.5762	0.6048	0.6102	0.6111
Hybrid-BERT-text	0.6003	0.6321	0.6023	0.6284	0.6322	0.6336
Hybrid-BERT-Rand-Row	0.6056	0.6356	0.6110	0.6294	0.6340	0.6350
Hybrid-BERT-Rand-Col	0.6105	0.6441	0.6094	0.6321	0.6388	0.6392
Hybrid-BERT-Rand-Cell	0.6124	0.6411	0.6117	0.6317	0.6381	0.6386
Hybrid-BERT-Cell-Mean	0.6104	0.6364	0.6148	0.6337	0.6385	0.6388
Hybrid-BERT-Cell-Max	0.6129	0.6410	0.6166	0.6349	0.6391	0.6395
Hybrid-BERT-Cell-Sum	0.6207	0.6473	0.6227	0.6397	0.6450	0.6454
Hybrid-BERT-Row-Mean	0.6196	0.6490	0.6216	0.6406	0.6456	0.6463
Hybrid-BERT-Row-Max	**0.6311**	**0.6673**†	**0.6361**	**0.6519**	**0.6558**	**0.6564**
Hybrid-BERT-Row-Sum	0.6199	0.6487	0.6168	0.6385	0.6436	0.6445
Hybrid-BERT-Col-Mean	0.6108	0.6395	0.6168	0.6340	0.6406	0.6412
Hybrid-BERT-Col-Max	0.6086	0.6324	0.6133	0.6297	0.6357	0.6362
Hybrid-BERT-Col-Sum	0.6131	0.6399	0.6131	0.6308	0.6384	0.6390

6.4.3 Baselines

We implement the following baseline methods:

- **Semantic Table Retrieval (STR)**: This is the method proposed by Zhang and Balag [299] which is the previous state-of-the-art method. It first represents queries and tables in multiple semantic spaces. Then multiple semantic matching scores are calculated based on the representations of queries and tables. Pointwise regression using Random Forest is used to fit those semantic features combined with other features. Like the original STR implementation, we set the number of trees to 1000 and the maximum number of features in each tree to 3.

- **Hybrid-BERT-text**: Only context fields are used and the table is not encoded except the table headers which are also considered as a context field.

- **Hybrid-BERT-Rand-Col**: Randomly selecting column items when constructing the BERT input.

- **Hybrid-BERT-Rand-Row**: Randomly selecting row items when constructing the BERT input.
- **Hybrid-BERT-Rand-Cell**: Randomly selecting cells from the table when constructing the BERT input.

For the BERT-based methods, we use the features proposed in [299] as v_a.

6.4.4 Experimental Results

We summarize our experimental results in Table 6.3. We can see that all BERT-based models can achieve better results than semantic table retrieval (STR). Even without encoding the tables, Hybrid-BERT-text can still outperform STR, which demonstrates that BERT can extract informative features from tables and context fields for ad hoc table retrieval. Randomly selecting columns, rows and cells have a marginal improvement on Hybrid-BERT-text, indicating that encoding the table content has the potential to further boost performance. In addition, the differences in performance among randomly selecting columns, rows and cells are not statistically significant. The answer to **RQ1** is very straightforward: all BERT based models with different content selection methods can perform better than the previous state-of-the-art method. Though the gain of performance is statistically significant at $p = 0.005$ level, BERT makes the main contribution, since only encoding context fields can achieve impressive results.

Next, we discuss the impact of item type and content selector. Comparing the results in Table 6.3, we observe that in general row item based methods are better than cell item based methods, and cell item based methods are better than column item based methods. Among all the methods, Hybrid-BERT-Row-Max achieves the best results across all metrics compared with all other methods. The improvement over all other methods is statistically significant at 0.05 level for MRR, and statistically significant at 0.05 level for NDCG@5, NDCG@15 and NDCG@20 except for Hybrid-BERT-Cell-Sum. It means that selecting rows that have the most significant signals is an effective strategy to construct BERT input within the length limit. In contrast to row items, column selection and cell selection based methods seem to be less effective. For several cases, content selection strategies for column and cell items even have worse performance than randomly selecting columns or cells. For example, Hybrid-BERT-Col-Max has MRR of 0.6324 while Hybrid-BERT-Rand-Col has MRR of 0.6441. Different from row items, max salience selector does not show superiority over other selectors for column items and cell items. It is expected that Hybrid-BERT-Rand-Col has better performance than Hybrid-BERT-Rand-Row, because a table is less likely to have more columns than rows, which means the probability of a potential optimal column to be selected is higher than that of a potential optimal row to be selected. For cell items, the sum salience selector shows marginally better performance than the other two selectors. And for column items, there is no clear best content selector but max salience selector seems to be the least effective.

Table 6.4: The setting of our methods where only BERT features are used.

Method Name	MAP	MRR	NDCG@5	NDCG@10	NDCG@15	NDCG@20
STR	0.5711	0.6062	0.5762	0.6048	0.6102	0.6111
BERT-text	0.5958	0.6240	0.5972	0.6206	0.6283	0.6287
BERT-Rand-Row	0.6005	0.6271	0.6063	0.6266	0.6310	0.6314
BERT-Rand-Col	0.6067	0.6400	0.6093	0.6327	0.6374	0.6380
BERT-Rand-Cell	0.6075	0.6358	0.6116	0.6287	0.6362	0.6369
BERT-Cell-Mean	0.6056	0.6331	0.6017	0.6274	0.6340	0.6343
BERT-Cell-Max	0.5967	0.6275	0.6013	0.6209	0.6299	0.6307
BERT-Cell-Sum	0.6149	0.6436	0.6151	0.6345	0.6420	0.6424
BERT-Row-Mean	0.6055	0.6365	0.6064	0.6314	0.6358	0.6363
BERT-Row-Max	**0.6277**	**0.6600**	**0.6274**	**0.6465**	**0.6517**	**0.6532**
BERT-Row-Sum	0.6113	0.6302	0.6077	0.6307	0.6356	0.6370
BERT-Col-Mean	0.6026	0.6318	0.6079	0.6269	0.6334	0.6339
BERT-Col-Max	0.6095	0.6398	0.6109	0.6319	0.6379	0.6385
BERT-Col-Sum	0.6059	0.6257	0.6050	0.6260	0.6339	0.6343

Three types of items are coherent units of the table with different granularities. A cell is the smallest unit compared with a row item or a column item. Usually, a column item is longer than a row item depending on the layout of the table. After manually examining some returned items, we find that cell item based methods are more biased towards returning items including query terms, while the methods based on the other two item types are forced to include some context information. Taking Figure 6.1c as an example, all the returned items include the term “wrestler” which appears in the rightmost column that includes a list of short biographies of professional wrestlers. However, for row items, other context information such as the names of the wrestlers are forced to be included. Since column items are usually longer than row items, if the content selector fails to return the most relevant column item as the first one, the model is less likely to achieve good performance. Based on our experiment results, we observe that max salience selector with row items has the best balance between accuracy and robustness, which answers **RQ3**.

6.5 Discussion

In this section, we continue the discussion of our proposed methods.

6.5.1 Ranking Only with BERT

To answer **RQ2**, we run the experiments that only use BERT features which means f equals f_{bert} in Equation 6.3. The results are shown in Table 6.4 where the method names correspond to the ones in Table 6.3 except the STR baseline and the prefix “Hybrid-” is removed. In all cases, performance decreases slightly when additional features are not used. In answer to **RQ2**, without additional features, all the proposed methods including baselines can outperform STR. Even without encoding table content, BERT-text can still achieve good performance which means the context fields are very important for ad hoc table retrieval. The conclusions are consistent with Section 6.4.4: sum salience selector is the best for cell items and max salience selector with row items still performs the best when only BERT features are used.

6.5.2 Generalization to Another Domain

Though we conclude that the max salience selector with row items is the best method, the conclusion may depend on the corpus. Therefore, we also conduct

Table 6.5: Results on WebQueryTable dataset.

Method Name	MAP
Feature + NeuralNet [242]	0.6718
BERT-Rand-Cell	0.6414
BERT-Row-Max	**0.7104**

experiments on a dataset from another domain. To do this, we use an open-domain dataset WebQueryTable[5] introduced by Sun et al. [242]. Unlike WikiTables where all the tables are from Wikipedia, the tables in WebQueryTable are collected from queried web pages returned by a commercial search engine. In total, 21,113 query-table pairs are manually annotated and the dataset is pre-split into training (70%), development (10%) and test (20%). In this scenario, no additional features are available for this corpus so only BERT features are used. Additionally, table caption, sub-caption and headers are used as context fields. The preprocessing is the same with WikiTables. We do not use the development set since we do not search for hyperparameters. We calculate the MAP scores of our models which are also reported by Sun et al. [242]. The results of the best BERT baseline method and the proposed method are shown in Table 6.5. The final results are also consistent with conclusions in Section 6.4.4—that max salience selector with row items is the best strategy.[6] Therefore, we can see that training BERT on row items with max salience selector is also an effective strategy for datasets in other domains, which makes the answers to **RQ2** and **RQ3** more convincing.

6.5.3 Feature-Based Approach of BERT

In Section 6.4.4, we use the fine-tuning approach that jointly fine-tunes the whole framework. In the experiment, we tried different methods to incorporate additional features. For example, we can directly concatenate additional features without any

[5]`https://github.com/tangduyu/Table-Intelligence/tree/master/table-search`

[6]We did not reproduce their method. We assume the results are comparable since the dataset is pre-split.

transformation with BERT features and feed the concatenated vector to the regression layer. We also tried to predict two relevance scores with BERT features and additional features separately, and then linearly transform them into a weighted relevance score. However, all of those variants perform worse than BERT-text. It is possible that BERT performance highly depends on the optimization strategy and adding other components for joint training can have negative impact on the fine-tuning process. To avoid such a case, we adapt BERT to a feature-based approach. First we use the fine-tuning approach to train BERT without additional features like in Section 6.5.1. Then we optimize the whole framework as in Section 6.4.4 except that BERT weights are not updated. The results are shown in Table 6.6. For the three item types, we only include the results of models using the best content selectors. All methods have significant improvements compared with fine-tuned approaches. Among the baselines, Hybrid-BERT-Rand-Col has the most improvement, which is even better than the best performance of BERT using content selectors for column items. Hybrid-BERT-Row-Max still achieves the best performance and the improvements over baselines are statistically significant at the level of $p = 0.005$.

So far, we observe that max salience selector with row items is the best strategy to construct inputs for BERT. In the feature-based approaches, it is more obvious that sum salience selector is the best one for cell items and mean salience selector is the best one for column items.

Table 6.6: Results using feature-based approaches. The superscript ‡ denotes statistically significant improvements over all baseline methods.

Method Name	MAP	MRR	NDCG@5	NDCG@10	NDCG@15	NDCG@20
Hybrid-BERT-text	0.6287	0.6546	0.6171	0.6489	0.6531	0.6536
Hybrid-BERT-Rand-Col	0.6590	0.6722	0.6481	0.6629	0.6692	0.6694
Hybrid-BERT-Rand-Row	0.6139	0.6418	0.6107	0.6345	0.6409	0.6411
Hybrid-BERT-Rand-Cell	0.6195	0.6554	0.6195	0.6382	0.6465	0.6466
Hybrid-BERT-Row-Max	**0.6737**‡	**0.7139**‡	**0.6633**‡	**0.6875**‡	**0.6924**‡	**0.6926**‡
Hybrid-BERT-Col-Mean	0.6379	0.6582	0.6229	0.6449	0.6540	0.6542
Hybrid-BERT-Cell-Sum	0.6643	0.6806	0.6529	0.6686	0.6739	0.6740

6.6 Analysis of BERT Features

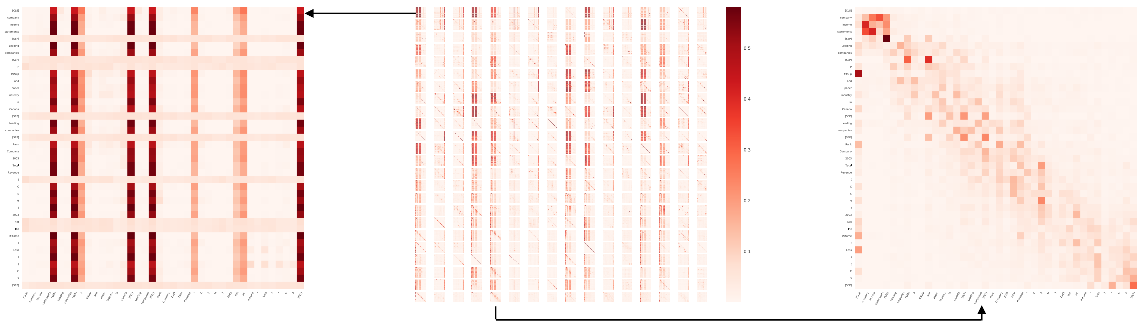


Figure 6.3: Middle figure includes all attention maps of a random test set example. Left figure shows the attention map of head 1 in the last layer. Similar attentions for [SEP] tokens result in the grid-look. Right figure shows the attention map of head 5 in the 1st layer with intra-sequence attention pattern. Attention weights with larger absolute values have darker colors.

Though BERT achieved new state-of-the-art results on various tasks, it is still unclear what are the exact mechanisms behind its success. In this section, we dive into the analysis of BERT for the table retrieval task. For illustration purposes, the results presented in this section are based on the weights of BERT-Row-Max. However, we observe similar patterns among different BERT-based methods and therefore the conclusions can also be applied to other methods.

6.6.1 Self-Attention Patterns

Compared to general scenarios where BERT is used for single-sequence or sequence-pair tasks, there are more than two sequences involved in the input of BERT for the table retrieval task and the sequence could have a lot of [SEP] tokens. BERT practitioners know [SEP] is a special token that is used as a delimiter of sequences. For our case, there could be a lot of [SEP] tokens in a single input and the number of [SEP] tokens are different across different samples. In this section we explore whether the self-attention patterns of BERT used in this chapter which involve multiple sequences are different from a BERT model fine-tuned on single sequence/sequence pair tasks.

We draw all the attention maps of a random example from the test set in Figure 6.3. We find all the types of self-attention maps categorized in Kovaleva et al. [128]: vertical, diagonal, vertical with diagonal, block and heterogeneous. We find that [SEP] embeddings in lower layers are attended or attending more differently than those in higher layers. Taking the 1st self-attention head in the 4th layer as an example, the 1st [SEP] embedding mainly attends to itself, while the other [SEP] embeddings mainly attend to [CLS] embedding. In contrast, the attentions for [SEP] tokens are very similar in higher layers resulting in a lot of grid-like attention maps (e.g., the left sub-figure of Figure 6.3). We also quantitatively measure the embeddings of different [SEP] tokens and calculate the smallest cosine similarity among all pairs of [SEP] in the same layer. The smallest cosine similarity is 0.78 in the 1st layer but increases close to 1 in higher layers, which means [SEP] tokens have different embeddings in lower layers, and after layers of self-attention, they have almost the same representations.

Besides the types of attention maps described by Kovaleva et al. [128], we observe some attention maps that look like scatter plots, which include sparse small blocks (e.g., head 1 in the 4th layer). This is because multiple sequences are included in a single input separated by [SEP] and some attention heads have a strong preference to put attention on multiple sequences (inter-sequence attention). We also observe there are self-attention heads that show intra-sequence attention patterns. For example, caption and section title both attend to themselves a lot in head 9 of the 1st layer. Query tokens attend a lot to themselves in head 5 of the 1st layer (right in Figure 6.3). The existence of **intra-sequence** and **inter-sequence attention patterns** may indicate that BERT can learn various sequence-level features through self-attention.

6.6.2 BERT Embedding Comparison

In the experiments, only the [CLS] embedding in the last layer is used as BERT features and the rest are not utilized. Here we further analyze the relationships

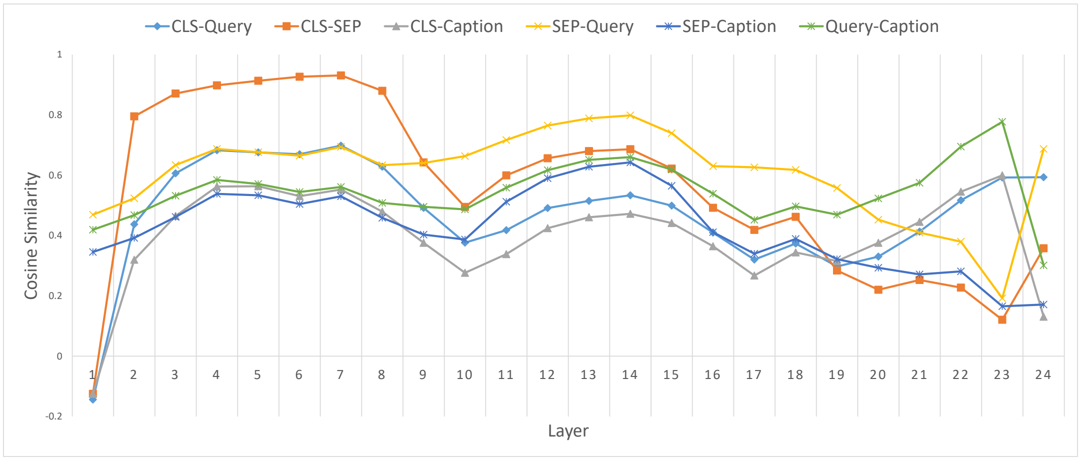


Figure 6.4: Average cosine similarity among different types of tokens in different layers.

among different types of BERT embeddings.

For each sample in the testing set, we extract embeddings corresponding to query tokens and average them as the query representation. We do the same for [SEP] and caption. Then we calculate the cosine similarity between every two of [CLS], [SEP], query and caption. We show the average cosine similarity of testing samples at different layers in Figure 6.4. We observe that the patterns between special/query tokens and table/context field tokens are similar, which means in Figure 6.4, if we replace caption with other context fields or selected items, the general patterns do not change. For example, "Query-Caption" is similar to "Query-Page title" and "CLS-Caption" is similar to "CLS-Page title".

In the 1st layer, [SEP] is close to query and caption while [CLS] is far from [SEP], query and caption. From layer 2 to layer 8, we note that [CLS] is very close to [SEP], which may indicate that [CLS] aggregates segment-level information through these layers. In contrast, the similarities among [SEP], query and caption do not change significantly from layer 2 to layer 14. It is interesting that from layer 23 to the last layer, query and [CLS] become closer but far away from caption. In the last layer, [SEP] is closer to query than [CLS], which may indicate [SEP] captures more query features than [CLS].

6.7 A New Test Collection for Web Table Retrieval

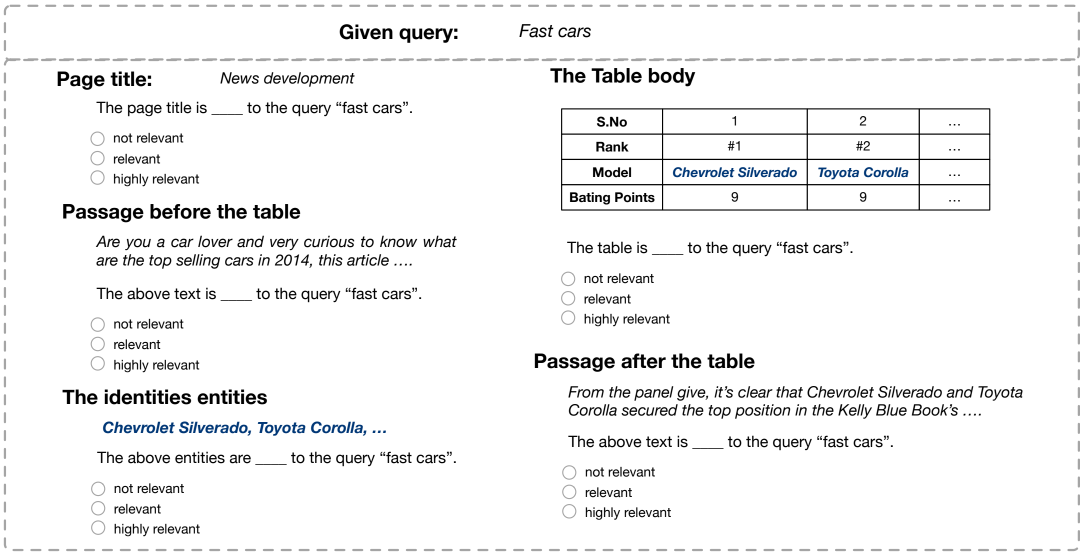
A Web table such as shown in Figure 6.1 usually has rich context information such as the page title and surrounding paragraphs. Prior datasets such as WikiTables [299] and WebQueryTable [242] only have a single relevance label for a dataset. However, the relevance of context fields can be different. In this section, we create and experiment with a new Web table retrieval (WTR) collection which has the following improvements compared with previous collections:

1. **Diversity.** The WikiTables collection [299] and GNQtables [234] only include tables from Wikipedia, while our collection covers broader topics retrieved from over 61,000 domains. Our collection can include any user-generated content and less than 1% of the tables are from Wikipedia. For the same query "Fast cars", the relevant tables in the WikiTables collection list facts about different car models, while tables in WTR can contain subjective information such as the example in Figure 6.5[7].

2. **Rich context.** As shown in Figure 6.5, each table in our new collection has four **context fields**: page title, text before the table, text after the table, and entities that are linked to the DBpedia [137]. We also keep other metadata about the source Web page. Details are in Section 6.7.2.

3. **Labels on multi-fields.** We notice that the relevance of a table and its context fields could be different. For example, the entities in Figure 6.5 are different car models and therefore can be considered as relevant to the query "Fast cars" while the page title "News development" is irrelevant. This is the first collection that has separate relevance labels for different sections of a Web page containing a table.

[7]Wikipedia has strict content policies (e.g., neutral point of view) and therefore is more limited to factual knowledge.



Given query: *Fast cars*

Page title: *News development*

The page title is ____ to the query “fast cars”.

- ◯ not relevant
- ◯ relevant
- ◯ highly relevant

Passage before the table

Are you a car lover and very curious to know what are the top selling cars in 2014, this article ....

The above text is ____ to the query “fast cars”.

- ◯ not relevant
- ◯ relevant
- ◯ highly relevant

The identities entities

Chevrolet Silverado, Toyota Corolla, ...

The above entities are ____ to the query “fast cars”.

- ◯ not relevant
- ◯ relevant
- ◯ highly relevant

The Table body

S.No	1	2	...
Rank	#1	#2	...
Model	*Chevrolet Silverado*	*Toyota Corolla*	...
Bating Points	9	9	...

The table is ____ to the query “fast cars”.

- ◯ not relevant
- ◯ relevant
- ◯ highly relevant

Passage after the table

From the panel give, it’s clear that Chevrolet Silverado and Toyota Corolla secured the top position in the Kelly Blue Book’s ....

The above text is ____ to the query “fast cars”.

- ◯ not relevant
- ◯ relevant
- ◯ highly relevant

Figure 6.5: Illustration of the crowdsourcing interface design.

4. **Reproducibility.** We describe the details of dataset pre-processing and release the run files of baseline methods for easy comparison.

The WTR collection contains not only 6,949 annotated query-table pairs but also query-context pairs for each context field which are ignored in previous test collections. We provide details of how the corpus is pre-processed (Section 6.7.1) and rankings of different table search methods (Section 6.7.4 and 6.7.5), making a future comparison with other work easier.

6.7.1 Constructing the Test Collection

In this section, we describe the test collection, including the table corpus, queries and the process of collecting relevance assessments.

Query and Table Corpus

We build the test collection with the English subset of WDC Table Corpus 2015[8] which includes 50.8M relational HTML tables extracted from the July 2015 Common

[8]http://webdatacommons.org/webtables/2015/EnglishStatistics.html

Crawl. All the tables are highly relational with an indexed core column including entities or a header row describing table attributes. Zhang et al. [302] further process the subset with novel entity discovery and link identified entities with their aliases to DBpedia. This results in 16.2M tables after retaining only those with at least one identified entity. Note that the WDC Table Corpus also includes Wikipedia pages (less than 1%) and can be considered as an extension of WikiTables collection [299]. However, this new collection contains tables from 61,086 different domain names and covers a broader range of topics.

The same set of queries with WikiTables collection [299] is used which includes 30 queries from Cafarella et al. [35] collected from Amazon's Mechanical Turk[9] platform and 30 queries from Venetis et al. [259] collected from query logs of searching for structured data.

Pooling

As a standard practice of IR test collection construction, we fetch the candidates by retrieving the top 20 tables using multiple unsupervised methods. Specifically, we use BM25 [222] to retrieve seven indexed fields: *Table*, *Caption*, *Page title*, *Header*, *TextBefore*, *TextAfter* and *Catchall*. The details of those fields are described in Section 6.7.2. The final assessment pool contains 6,949 query-table pairs.

Collecting Relevance Judgments

In the WikiTables collection, a table, along with other context fields such as page title and section title, is presented to an annotator, while only a single relevance label is assigned. However, a table does not always have the same relevance label as context fields. For example, the entities in Figure 6.5 representing different car models are relevant to the query "Fast cars" while the page title "News development" is irrelevant. Therefore, for each record (i.e., query-table pair) in our new collection, we ask the annotator to judge the relevance of each field concerning a query as shown

[9]`https://www.mturk.com/`

in Figure 6.5. We use Amazon's Mechanical Turk to collect all the judgments based on a three-point scale:

- **Irrelevant (0)**: this field is irrelevant to the query (i.e., based on the context you would not expect this to be shown as a result from a search engine).
- **Relevant (1)**: this field provides relevant information about the query (i.e., you would expect this Web page to be included in the search results from a search engine but not among the top results).
- **Highly relevant (2)**: this field provides ideal information about the query (i.e., you would expect this Web page ranked near the top of the search results).

To control the annotation quality, 135 query-table pairs for one query annotated by Chen et al. [59] are taken as testing questions (where at least two of the experts agreed on the relevance label). Each of the remaining query-table pairs is paired with a randomly sampled testing question, which means a single assignment consists of two query-table pairs. Based on the testing questions, we consider the following metrics:

- **Assignment accuracy**: For an assignment, if 3 out of 5 relevance judgments corresponding to gold labels of a testing question are correct, then the assignment accuracy is 60%.
- **Worker accuracy**: If a worker submits the results of 5 assignments, there are $5 \times 5 = 25$ labels of 5 testing questions. If 20 out of the 25 labels are correct, then the worker accuracy is 80%.
- **Approval rate**: If a worker submits 50 assignments and 30 assignments are approved, then the approval rate of the worker is 60%.

We first approve those assignments with at least 60% assignment accuracy or the assignments from workers whose worker accuracy is at least 80%. Then we only allow those workers whose approval rates are at least 50% to continue the remaining annotation process. We collected 3 judgments for each record and paid workers

5 cents per assignment. In Table 6.7, we show the Fleiss' Kappa inter-annotator agreement of different sections. From the table, we can see that the annotators disagree with the judgments on entities the most and make fair agreements on other sections. To determine the relevance label of each section to a query, we took the majority vote among three judgments. In case of a tie, we took the average of relevance scores as the final judgment. We compare the statistics of labels between WTR and WikiTables collection [299] in Figure 6.6. We can see that the proportion of 0,1,2 labels in the two datasets are similar but our collection is more than twice the size of the WikiTables collection.

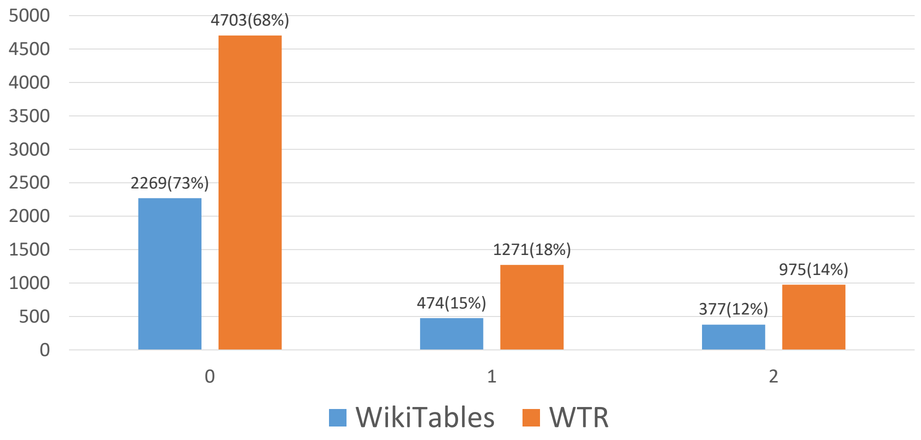


Figure 6.6: Comparison of relevance label distributions between WikiTables and the WTR collection.

Table 6.7: The Fleiss' Kappa inter-annotator agreement of different sections.

Field	Fleiss' Kappa
page title	0.28
text before the table	0.26
table	0.27
entities	0.20
text after the table	0.23

Inter-field analysis

One of our main contributions is that we provide the relevance annotations of context fields of a table. The authors of WikiTables collection [299] only asked workers to assign a single label for a table, but the context information (e.g., page title) was also presented to workers, which may mislead the workers. For example, if a page title is relevant while the table is not, then a worker may still consider it as relevant. Besides, lack of context could lead to misunderstanding of tables. For example, the table in Figure 6.5 provides a ranked list of cars. Without surrounding passages saying the ranking criteria are subjective ratings from customers, an annotator may think the cars in the table are ranked by speed and annotates it as relevant to the query "fast cars".

To study the discrepancy of relevance judgments among different fields, we could make the assumption that the four relevance labels of context fields are also relevance judgments for the Web table. It resembles the situation in which we ask five annotators to judge every query-table pair and we want to know the agreement between any two raters (fields). The Cohen's Kappa statistics between any two raters (fields) are summarized in Table 6.8. As we can see from the table, different context fields have different levels of agreement with *Table*. Among all the context fields, *TextAfter* has the most agreement with *Table* while *Page Title* has the least agreement with *Table*. *Page Title* often includes little information and even if its content is relevant the worker can hardly recognize it. However, the content in *TextAfter* is more likely to be the paragraph that explains the details of the *Table* which deepens the reader's understanding. As shown in Figure 6.5, *Page Title* "News Developments" seems to be a general title of a news website and tells nothing related to the query or *Table* on the same page. But the content in *TextAfter* further illustrates the *Table*.

Table 6.8: The Cohen's kappa inter-annotator agreement between any two fields.

	TextBefore	Page title	Table	Entities	TextAfter
TextBefore	1	0.44	0.41	0.34	0.41
Page title	0.44	1	0.39	0.33	0.39
Table	0.41	0.39	1	0.4	0.43
Entities	0.34	0.33	0.4	1	0.37
TextAfter	0.41	0.39	0.43	0.37	1

6.7.2 Preprocessing and Indexing

Leveraging the entity linking results of Zhang et al. [302] which are formed as ¡table mention, entity entries¿ pairs[10], we construct the table corpus by extracting the original tables from the English subset of WDC Table Corpus 2015[11] and mapping mentions to KB entries. This corpus is comprised of 3M relational tables. In addition to the original fields (cf. Table 6.9), each table has an *Entities* field, listing all the in-KB entities identified by Zhang et al. [302]. We use Elasticseach[12] for indexing the tables, separated by the fields in Table 6.9.

The scripts to download, preprocess and index the data are also available on our GitHub repository.[13]

6.7.3 Compared Methods

We compare with both unsupervised and supervised methods.

- **Single-field document ranking:** Each table is represented as a single document [36]. In our corpus, the content in the *Catchall* field described in Table 6.9 is used as the table representation. Then classic IR methods like Language Models with Dirichlet smoothing are used for ranking.

[10]https://zenodo.org/record/3627274#.YC91NehKh1Q

[11]http://data.dws.informatik.uni-mannheim.de/webtables/2015-07/englishCorpus/compressed/

[12]https://www.elastic.co/downloads/past-releases/elasticsearch-5-3-0

[13]https://github.com/Zhiyu-Chen/Web-Table-Retrieval-Benchmark

Table 6.9: Table fields used when constructing the corpus.

Field	**Definition**
Page title	Title of the Web page where the table lies. It is determined from HTML tags
TextBefore	200 words before the table
Caption	Caption of the table
Table	The extracted table from a Web page
Header	Headers of the table if exist
Entities	Entities in the tables identified by the novel entity discovery process [302]
TextAfter	200 words after the table
Orientation	Orientation can be either horizontal or vertical. A horizontal table has attributes in columns while the attributes of a vertical table are represented in rows
URL	The original web address of the Web page
Key Column	For a horizontal table, the key column is the one contains the names of the entities
Catchall	The concatenation of page title, caption, table, text before and after the table

- **Multi-field document ranking:** Each table is represented as a multi-field document [200]. We use the following five fields: *page title*, *TextBefore*,*Table*, *TextAfter* and *Header*. Unlike [200] and [299], we do not consider *Caption*. Since we find that few tables have non-empty table captions and most tables with captions are from Wikipedia.

- **LTR:** **L**earning-**T**o-**R**ank [299] is a feature-engineering approach leveraging table structure and lexical features (Features 1-12 in Table 6.10). A random forest is used to fit the ranking features in a pointwise manner.

- **STR:** The **S**emantic-**T**able-**R**etrieval approach [299] extends LTR with semantic features such as bag-of-categories, bag-of-entities, word embeddings, and graph embeddings. These embeddings are fused in different strategies [297] to generate ranking features (Features 13-15 in Table 6.10). Like LTR, a random forest is used for pointwise regression.

We compare the above methods with BERT-ROW-MAX proposed in this chapter.

Table 6.10: Features extracted from Web tables which are used in LTR and STR methods.

ID	Description	Dim.
1	Number of query terms	1
2	Sum of query IDF scores (from indexed fields except *Caption*)	6
3	The number of rows in the table	1
4	The number of columns in the table	1
5	The number of empty table cells	1
6	Ratio of table size to page size	1
7	Total query term frequency in the leftmost column cells	1
8	Total query term frequency in second-to-leftmost column cells	1
9	Total query term frequency in the table body	1
10	Ratio of the number of query tokens found in page title to total number of tokens	1
11	Ratio of the number of query tokens found in table title to total number of tokens	1
12	Language modeling score between query and multi-field document representation of the table	1
13	Four semantic similarities between the query and table represented by bag-of-word embeddings.	4
14	Four semantic similarities between the query and table represented by bag-of-entities.	4
15	Four semantic similarities between the query and table represented by bag-of-graph embeddings.	4

6.7.4 Implementation Details

For single-field and multi-field document ranking, we use the implementations from Nordlys [103] which has interfaces to Elasticsearch. Since the features used by LTR

Table 6.11: The evaluation results of compared table retrieval baseline methods.

Model	**MAP**	**P@5**	**P@10**	**NDCG@5**	**NDCG@10**
Single-field document ranking	0.5071	0.4380	0.3966	0.4124	0.4851
Multi-field document ranking	0.5104	0.4407	0.3943	0.4201	0.4916
LTR	0.5878	0.5300	0.4433	0.5313	0.5870
STR	0.6210	0.5493	0.4727	0.5585	0.6258
BERT-ROW-MAX(base)	**0.6346**	**0.5713**	**0.4800**	**0.5737**	**0.6327**

Table 6.12: Table retrieval evaluation results with different annotation "bias". We highlight the cases where the results are improved over the original method.

Model	**MAP**	**P@5**	**P@10**	**NDCG@5**	**NDCG@10**
STR	0.6210	0.5493	0.4727	0.5585	0.6258
STR (Max-Entities)	**0.6261 (+0.82%)**	**0.5573 (+1.46%)**	**0.4763 (+0.77%)**	**0.5634 (+0.88%)**	**0.6281 (+0.36%)**
STR (Min-Entities)	0.5893 (-5.11%)	0.5253 (-4.37%)	0.4533 (-4.10%)	0.5251 (-5.99%)	0.5872 (-6.18%)
STR (Max-PageTitle)	0.6137 (-1.18%)	0.5487 (-0.12%)	0.4677 (-1.06%)	0.5517 (-1.22%)	0.6154 (-1.67%)
STR (Min-PageTitle)	0.6060 (-2.42%)	0.5473 (-0.36%)	0.4613 (-2.40%)	0.5419 (-2.98%)	0.6041 (-3.47%)
STR (Max-TextBefore)	0.6127 (-1.33%)	0.5380 (-2.06%)	0.4703 (-0.50%)	0.5472 (-2.03%)	0.6171 (-1.39%)
STR (Min-TextBefore)	0.6021 (-3.04%)	0.5480 (-0.24%)	0.4547 (-3.81%)	0.5471 (-2.04%)	0.6003 (-4.08%)
STR (Max-TextAfter)	0.6203 (-0.12%)	**0.5533 (+0.73%)**	**0.4770 (+0.91%)**	0.5545 (-0.72%)	**0.6265 (+0.11%)**
STR (Min-TextAfter)	0.6007 (-3.26%)	0.5327 (-3.03%)	0.4577 (-3.17%)	0.5294 (-5.21%)	0.5970 (-4.61%)

rely on the source of tables and some features extracted from Wikipedia tables are not available for Web tables (e.g., number of page views), we generate the features which apply to the new test collection. The original STR calculates four semantic representations for queries and tables. Since there are no Wikipedia categories for all Web tables, we use three semantic representations for queries/tables: bag-of-entities, word embeddings, and graph embeddings. For each type of semantic representation, we use early fusion, late-max, late-sum, and late-max, as described in Zhang et al. [297] to obtain four semantic matching scores, which results in 12 semantic matching features in total for each query-table pair. We summarize the features in Table 6.10, where features 1-12 are used in LTR and features 1-15 are used in STR. The scikit-learn[14] implementation of random forest is used for both LTR and STR. We set the number of trees to 1000 and a maximum number of features in each tree to 3. For BERT-ROW-MAX, we use the pre-trained BERT-base-cased

[14]`https://scikit-learn.org/`

model[15] which consists of 12 layers of Transformer blocks. As in Section 6.4, we train the model with 5 epochs, and batch size is set to 16. The Adam optimizer [123] with a learning rate of 1e-5 is used to optimize BERT-ROW-MAX. A linear learning rate decay schedule with a warm-up ratio of 0.1 is also used. We train all the models using 5-fold cross-validation (w.r.t. NDCG@5). To facilitate fair comparison in future work, we prepared the five data splits used for cross-validation in our WTR repository[13], which is not provided by previous work [299].

6.7.5 Overall Performance

In Table 6.11 we report on the performance of different table retrieval methods. The conclusion is consistent with Section 6.4 and Section 6.5 that BERT-ROW-MAX achieves the best overall evaluation metrics. STR outperforms LTR and other unsupervised methods significantly. However, different from the results on WikiTables collection, BERT-ROW-MAX does not outperform STR sufficiently. We speculate that the new corpus may include more noise which makes it more difficult for neural models to learn features. In contrast, LTR and STR are trained on curated features which are more robust to noise in raw text.

6.7.6 Utilizing Labels of Context Fields

Recall that we have explicitly collected the relevance judgments of different fields in Section 6.7.1, we observe that a context field does not always have the same relevance label as the table. This discrepancy could result in bias in the annotation process of WikiTables collection. For example, a strict annotator may think a table should be annotated as relevant when all the context fields are also relevant. While a lax annotator may consider a table as relevant when any of the context fields provides related information even if the table itself is not relevant. To investigate how the labels of context fields can help table retrieval, e.g., how the annotation bias could potentially affect the models, we employ two strategies to resemble a strict

[15] `https://huggingface.co/transformers/pretrained_models`.html

annotator and a lax annotator: given the original label l_t of a table and the label l_c of one context field, we re-annotate l_t as either $max(l_t, l_c)$ or $min(l_t, l_c)$. Then we can use the "new" labels to train the models but evaluate on original labels.

We take the STR approach as the example and in total form 8 variations for STR which are trained with "biased" labels by combining the min/max strategy with four context fields. We summarize the results of STR trained on new labels in Table 6.12. For example, STR (max-entity) means STR is trained with the new labels where the maximum relevance among a table and its *Entities* field is used as the training label for each query-table pair. Note that we only change the labels in the training set and keep the original testing set since the task is still table retrieval rather than context field retrieval. We can observe that the performance of majority models decreases compared with the model trained on the original labels, which demonstrates that the biased annotation can harm the model training. Among those cases, STR (Min-Entities) has the most performance drop. Nevertheless, there are a few cases where the performance slightly increased. It is worth noting that STR (max-entity) performs better than the model trained on the original labels on all the evaluation metrics. We observe the same results on other supervised methods. Table retrieval and entity retrieval may be highly synergistic tasks. Taking good advantage of labels from *Entities* can help table retrieval while misusing them can lead to a large performance drop.

6.8 Summary

We have addressed the problem of ad hoc table retrieval with the deep contextualized language model BERT. Considering the structure of a table, we propose three content selectors to rank table items in order to construct input for BERT which effectively utilize useful information from tables and overcome the input length limit of BERT to some extent. We combine BERT features and other tables features to solve the table retrieval task as a pointwise regression problem. Our proposed Hybrid-BERT-Row-Max method outperforms the previous state-of-the-art and BERT baselines with a large margin on the WikiTables dataset. Through empirical experiments, we find that using the max salience selector with row items is the best strategy to construct BERT input. Overall, we also find that sum salience selector is the best for cell items. While for column items, the mean salience selector only seems to be the best when a feature-based approach is used. We further show that the feature-based approach of BERT is better than jointly training BERT with a feature fusion component. We also conduct experiments on the WebQueryTable dataset and demonstrate that our method generalizes to other domains.

Our analysis on fine-tuned BERT shows that various sequence-level features are captured by the self-attention of BERT and [CLS] embedding tends to aggregate sequence-level information, which could explain why using it as features is effective for the ad hoc table retrieval task. We also find that [SEP] embeddings from the last layer of BERT are very close to query embeddings, which suggests that making use of [SEP] has the potential to further improve performance.

We also propose a new test collection WTR for Web table retrieval. Compared with previous datasets, WTR covers a broader range of topics and includes tables from over 61,000 different domain names. Since a Web table usually has rich context information such as the page title and surrounding paragraphs, we not only provide relevance judgments of query-table pairs, but also the relevance judgments of query-table context pairs with respect to a query, which are ignored by previous test collections. In the experiments, we show that utilizing the labels from context fields may be helpul for table retrieval.

6.9 Bibliographic Notes

Given the advances of deep contextualized language models for natural language understanding tasks, researchers from the IR community have begun to study BERT for IR problems. Nogueira et al. [184] describe an initial application of BERT for passage re-ranking task where the sentence-pair classification score is used. Nogueira et al. [185] then propose a multi-stage document ranking framework where BERT is used for pointwise and pairwise re-ranking. Yang et al. [285] show that treating social media text retrieval as a sentence pair classification task can achieve new state-of-the-art results. Then they apply BERT to a dataset with longer documents and rank a document with linear interpolation of the original document score and weighted top-n sentence scores. Similarly, Dai et al. [68] use passage-level evidence to fine-tune BERT and consider all passages from a relevant document as relevant. They first predict the relevance score of each passage independently. The document relevance is the score of the first passage, the best passage, or the sum of all passage scores. BERT has also been applied to FAQ retrieval task by Sakata et al [226] where given a user query, a question is scored by the combination of question-question BM25 score and question-answer BERT score. MacAvaney et al. [159] combine the BERT classification token with existing neural IR models. The experiments show that this joint approach can outperform a vanilla BERT ranker.

IR researchers have also investigated the possible reasons why BERT can have such substantial improvements for IR problems. Through carefully designed experiments, Padigela et al. [192] show that BM25 is more biased towards high-frequency terms which hurt its performance while BERT has a better ability to discover the semantic meaning of novel terms in documents with respect to query terms. They also find that BERT has less performance improvement compared with BM25 for long queries. Dai et al. [68] demonstrate that BERT can take advantage of stopwords and punctuation in the queries which is in contrast to traditional IR models. Qiao et al. [206] show that BERT can be categorized into interaction-based IR models because simply obtaining query and document representations from BERT independently and then computing their cosine similarity results in performance close to random. They also find that BERT assigns extreme matching scores to query-document pairs and most pairs get either one or zero ranking scores.

Many researchers (e.g., [185, 285, 68, 163]) find that the length limit of BERT causes difficulties in training. Mass et al. [163] specifically study the effect of passage length and segmentation configurations on passage re-rank performance. They find that mid-sized (256 tokens) inputs achieve the best results for the selected datasets.

Dai and Callan's method [68] to deal with long documents as mentioned before may result in noisy positive samples because for a relevant document, not all sentences are relevant to a query. The splitting and then aggregating methods in these approaches can increase the training and inference cost several times. In this chapter, we pre-select the segments from the input with low-cost methods and then use BERT for the downstream table retrieval task.

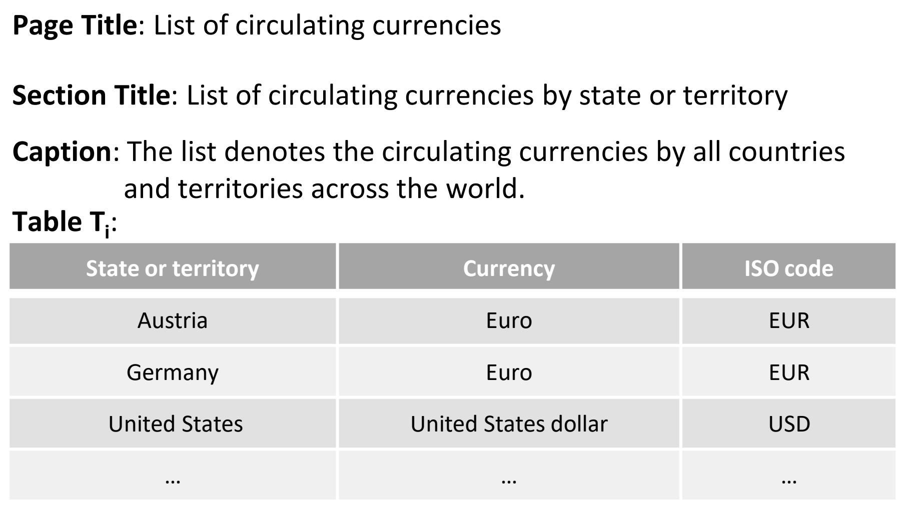
Chapter 7

Dataset Search with Graph Neural Networks

Table retrieval methods in previous chapters only consider the textual information of tables and the structural information is rarely used. In this chapter, we propose to model the complex relations in the table corpus as one or more graphs and then utilize graph neural networks to learn representations of queries and tables. We show that text-based table retrieval methods can be further improved by graph-based predictions which fuse multiple field-level information.

7.1 Introduction

A massive number of tables extracted from the Web have been used in various research tasks such as question answering [52], entity linking [27, 302] and table augmentation [26, 71, 54, 287]. Previous table search methods [299, 253, 55, 57, 249, 59] either treat a table as a regular or multifield document. The structure of a table is underutilized. Chen et al. [57] slice a table into smaller pieces and select only the most salient ones for final ranking. However, the structure information across different tables is missed. As a structured document, a table itself can be naturally viewed as a graph. The words appearing in the same column or row usually have certain relationships. Such relationships can hardly be captured by models which treat the table as flat text. As shown in Figure 7.1, we know that "Euro" and "United States Dollars" are two different types of currencies since they appear in the same column "Currency". If the table is flattened into a sequence, the existing methods [200, 299, 57] for text retrieval will neglect the semantic relationship between table attributes/headers and table values. The only structure information after flattening operation is the order of tokens. However, this artificially-created order information is not a part of the original data and could mislead the model training.



Page Title: List of circulating currencies

Section Title: List of circulating currencies by state or territory

Caption: The list denotes the circulating currencies by all countries and territories across the world.

Table T_i:

State or territory	Currency	ISO code
Austria	Euro	EUR
Germany	Euro	EUR
United States	United States dollar	USD
…	…	…

Figure 7.1: An example of a Web table with page title and section title as context fields.

In this chapter, we propose a graph neural network-based method for ad hoc table retrieval. By explicitly modeling the table corpus as one or more graphs, we directly encode the structural information of the table which is often ignored by previous methods. Our principal contributions are:

1. We propose a novel multi-graph neural network method for ad hoc table retrieval.

2. We propose table-based pointwise mutual information (TPMI) to calculate the semantic correlation of terms in the table corpus.

3. The experimental results demonstrate that our proposed method outperforms baselines and can improve the performance of previous sequence-based methods.

7.2 Methods

In this section, we introduce the details of our proposed ***M**ulti-**G**raph **NE**ural networks for **T**able **S**earch* (MGNETS for short). Our method includes four modules: graph construction, multi-graph neural network encoder, query/table pooling layer and ranking layer. First, we describe how to construct graphs from the table corpus (Section 7.2.2). Then we propose to use graph neural networks as a encoder to learn node representations through message passing (Section 7.2.3). After that, we propose different ways to learn query/table representations from node embeddings (Section 7.2.4). In the end, the ranking layer predicts the final relevance score with query/table representations as input (Section 7.2.5).

7.2.1 Problem Statement

In the task of table search, given a query q usually consisting of several keywords $q = \{k_1, k_2, ..., k_l\}$, our goal is to rank a set of tables $D = \{T_1, T_2, ..., T_n\}$ in descending order of their relevance scores with respect to q. Each table T_i has corresponding context fields $\{p_{i1}, ..., p_{ik}\}$. A data table is a set of cells arranged in rows and columns like a matrix. Each cell could be a single word, a real number, a phrase or even sentences. The first row of a table is the header row and consists of header cells. The context fields associated with a table instance depend on the source of the dataset. For example, a table from Wikipedia usually has caption, page title, and section title as context fields as shown in Figure 7.1.

7.2.2 Graph Construction

To construct the graph $\mathcal{G}$ for table search, we first need to define the nodes set $\mathcal{V}$ and edges set $\mathcal{E}$ of the graph. We have two types of nodes in the graph. Every unique term that appears in the data tables or context fields is represented as a **term node** $v_t \in \mathcal{V}$. Every data table has a corresponding **table node** $v_T \in \mathcal{V}$. We add an edge between two nodes if they have a co-occurrence relationship. We list the possible types of edges below.

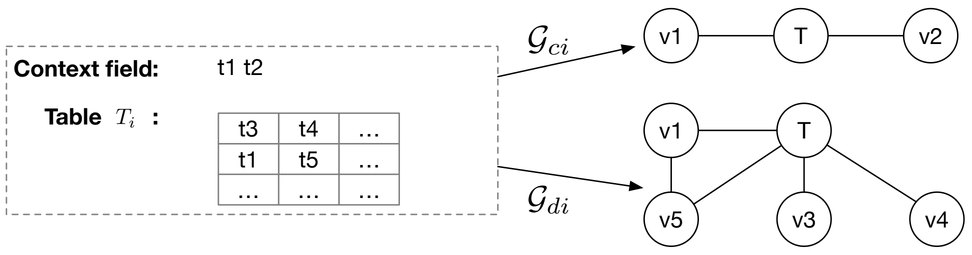


Figure 7.2: An example of constructed graphs from a table.

Table-Term Edges. A term node can be from either a table or a context field of a table. A table-term edge $(v_T, v_t) \in \mathcal{E}$ is constructed if the term t occurs in the table T or any context fields of table T. We can also treat queries in the training set as another context field. However, in experiments (not shown) we find our method can still achieve good performance without constructing edges and nodes from queries.

Term-Term Edges. In previous work of applying graph neural networks for text classification [286, 150], a fixed size sliding window is applied to all documents in the corpus and the pointwise mutual information (PMI) between two terms is calculated to determine the corpus-level co-occurrence. Unlike a text document, a data table has a non-linear structure so that defining the co-occurrence relationship using a fixed size sliding window and calculating the PMI is not directly applicable. Moreover, the cross-field co-occurrence is undefined in previous methods. We can treat every field as a document and then calculate the PMI score for term pairs in different fields. However, the co-occurrence of a term pair in one field does not indicate its co-occurrence in another field. In fact, the context fields have short lengths as shown in Figure 7.1. Besides, lots of context fields are single phrases describing the topic of the corresponding data table, which indicates that the relationships between tables and context fields (already described by table-term edges) are more important than the intra-context fields relations. Instead of using a fixed size sliding window, we treat every column or every row as the sliding window. For a data table with r_i rows and c_i columns, there are $r_i + c_i$ context windows. Now we define the **table-based pointwise mutual information (TPMI)** score for term pair t_i and t_j:

$$\begin{aligned} TPMI(t_i, t_j) &= log\frac{P(t_i, t_j)}{P(t_i)P(t_j)} \\ &= log\frac{C(t_i, t_j)/C}{C(t_i)/C \times C(t_j)/C} \\ &= log\frac{C(t_i, t_j) \times C}{C(t_i)C(t_j)} \\ C &= \sum_{i=1}^{n} r_i + c_i \end{aligned} \tag{7.1}$$

where $C = \sum_{i=1}^{n}(r_i + c_i)$ corresponding to the total number of sliding windows in all data tables, $C(t_i)$ is the number of rows and columns containing t_i and $C(t_i, t_j)$ is the number of rows and columns containing both t_i and t_j. A positive TPMI score indicates two terms are semantically correlated. We add (t_i, t_j) into $\mathcal{E}$ if two table terms t_i and t_j have a positive TPMI score.

Multi-graph Construction. We can construct one heterogeneous graph which contains all types of edges mentioned above. However, the semantic relation of a term pair in one field does not imply the relation in another field and constructing a graph with all possible edges could result in ambiguous semantic meanings. Therefore, we construct multiple subgraphs and each subgraph captures certain semantic relations among nodes. In this chapter, we construct two subgraphs $\mathcal{G}_d = (\mathcal{V}_d, \mathcal{E}_d)$ and $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$:

$$\begin{aligned} \mathcal{V}_d &= \{\mathcal{V}_T \cup \mathcal{V}_{dt}\} \qquad\qquad \mathcal{V}_c = \{\mathcal{V}_T \cup \mathcal{V}_{ct}\} \\ \mathcal{E}_d &= \{(v_i, v_j)|(v_i, v_j) \in \mathcal{E}, v_i, v_j \in \mathcal{V}_d\} \\ \mathcal{E}_c &= \{(v_i, v_j)|(v_i, v_j) \in \mathcal{E}, v_i, v_j \in \mathcal{V}_c\} \end{aligned}$$

where $\mathcal{V}_T$ is the set of nodes representing data tables, $\mathcal{V}_{dt}$ represents all term nodes in data tables, and $\mathcal{V}_{ct}$ represents the nodes constructed from context fields. Note that both subgraphs have table nodes and in each subgraph two table nodes are not directly connected but indirectly connected by co-occurred terms. The first subgraph contains term nodes from data tables while the second subgraph contains term nodes from context fields. By building such a heterogeneous graph, both inter-field and intra-field information are captured. An example of constructed graphs $\mathcal{G}_{ci}$ and $\mathcal{G}_{di}$ from table T_i is shown in Figure 7.2. $\mathcal{G}_d$ can be constructed by merging all $\mathcal{G}_{di}$ for $T_i \in D$. $\mathcal{G}_c$ can be constructed in a similar way.

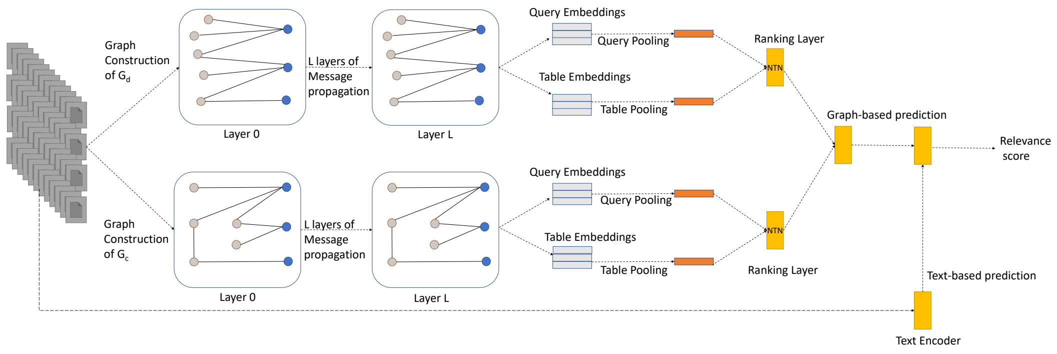


Figure 7.3: An illustration of the overall framework. From the corpus we construct two different graphs. After message propagation through the multi-graph encoder, we learn query and table representations from each graph. We obtain the final ranking score from graph-based prediction and also the text-based prediction from a text encoder.

7.2.3 Multi-graph Encoder

After obtaining the constructed graphs $\mathcal{G}_d$ and $\mathcal{G}_c$, we can employ graph neural networks to learn the embeddings of nodes. The representation learning process of GNN models can be divided into two steps: neighborhood aggregation and combination [281]. The k-th layer of a GNN model is:

$$h_v^{(k)} = \phi^{(k)}(h_v^{(k-1)}, \psi^{(k)}(\{h_u^{\{k-1\}} : u \in \mathcal{N}(v)\})) \tag{7.2}$$

where $h_v^{(k)}$ denotes the feature for node $v \in \mathcal{V}$ at k-th layer, $\mathcal{N}(v)$ denotes the neighborhood nodes of $v \in \mathcal{V}$. $\psi^{(k)}$ is the aggregation function at k-th layer which aggregates the representations of v's neighbors. $\phi^{(k)}$ generates the node representation of v at the k-th layer by combining its representation from the previous layer and aggregated neighbor representations. For different GNN models, the aggregate and combine functions are different [124, 230, 257]. In this chapter, we use Graph Isomorphism Network (GIN) [281] as the multi-graph encoder to learn node embeddings in $\mathcal{G}_d$ and $\mathcal{G}_c$:

$$h_d^{(k)} = MLP_d^{(k)}((1+\epsilon_d^{(k)})h_d^{(k-1)} + \sum_{u \in \mathcal{N}(v_d)} h_u^{(k-1)}) \tag{7.3}$$

$$h_c^{(k)} = MLP_c^{(k)}((1+\epsilon_c^{(k)})h_c^{(k-1)} + \sum_{u \in \mathcal{N}(v_c)} h_u^{(k-1)}) \tag{7.4}$$

where $\epsilon_d^{(k)}$ and $\epsilon_c^{(k)}$ are learned parameters. Multi-layer perceptrons (MLPs) are used to obtain features at k-th layer from aggregated node embeddings. We initialize the node embeddings $h_d^{(0)} \in \mathbb{R}^{|\mathcal{V}_d|}$ and $h_c^{(0)} \in \mathbb{R}^{|\mathcal{V}_c|}$ with one-hot encoding features for $v_d \in \mathcal{V}_d$ and $v_c \in \mathcal{V}_c$ respectively. After stacking l layers of GNN models, we obtain $l+1$ embeddings for each node in $\mathcal{G}_d$ and $\mathcal{G}_c$.

7.2.4 Pooling Layer

The community has been studying how to design readout functions at the node level for node classification and graph level for graph classification [124, 230, 257, 281]. In this section, we propose to learn query and table representations from node embeddings obtained from Section 7.2.3. We generate two representations for queries and two for tables: one representation is based on $\mathcal{G}_d$ and the other is based on $\mathcal{G}_c$.

Given a query table pair (q, T_i) and $\mathcal{G}_d$, we can find the set of nodes $\mathcal{V}_q \subseteq \mathcal{V}_d$ and $\mathcal{V}_{ti} \subseteq \mathcal{V}_d$ constructed from query $q \in Q$ and table T_i respectively. To learn the representation of q and T_i in $\mathcal{G}_d$, we can use the graph-level readout function which has been previously used in graph classification. However, to learn the representation of a query or a table at k-th layer, we operate on the subsets of nodes instead of all the nodes:

$$h_{qd}^{(k)} = Sum(\{h_v^{(k)} : v \in \mathcal{V}_q\}), \quad h_{ti}^{(k)} = Sum(\{h_v^{(k)} : v \in \mathcal{V}_{ti}\}) \tag{7.5}$$

In experiments not shown here, we find that summation of node embeddings is more effective than mean pooling or max pooling. Equation (7.5) does not require additional parameters which makes it computationally efficient. Therefore, we use it for the initial node embeddings which are one-hot encoding features. Inspired by SimGNN [16], we use the following attention-based pooling function to learn query/table representations for other layers ($k > 0$):

$$h_{qd}^{(k)} = {H_{qd}^{(k)}}^\top \cdot \sigma(H_{qd}^{(k)} \cdot Mean(H_{qd}^{(k)} W_1^{(k)})) \tag{7.6}$$

$$h_{ti}^{(k)} = {H_{ti}^{(k)}}^\top \cdot \sigma(H_{ti}^{(k)} \cdot Mean(H_{ti}^{(k)} W_1^{(k)})) \tag{7.7}$$

where $H_{qd}^{(k)} \in \mathbb{R}^{|\mathcal{V}_q| \times d}, H_{ti}^{(k)} \in \mathbb{R}^{|\mathcal{V}_{ti}| \times d}$ denotes node embeddings of the query and the table respectively; $Mean(\cdot)$ calculates the average embedding after the node embeddings are transformed by a trainable weight $W_1^{(k)} \in \mathbb{R}^{d \times d}$. Note that $W_1^{(k)}$ is a shared parameter. $\sigma(\cdot)$ is the sigmoid function and its output can be considered as the weights for query and table nodes so that the final representation is the weighted sum of all node embeddings.

Given the context fields of T_i and $q \in Q$, we obtain the query representation $h_{qc}^{(k)}$ and context field representation $h_{ci}^{(k)}$ from $\mathcal{G}_c$ in a similar way based on Equations (7.5 - 7.7).

7.2.5 Ranking Layer

In Section 7.2.4, we generate the query representation ($h_{qd}^{(k)}$ and $h_{qc}^{(k)}$), table representation ($h_{ti}^{(k)}$) and context field representation ($h_{ci}^{(k)}$) from each layer of the multi-graph encoder with a pooling layer. In order to predict the final relevance score of a given query table pair (q, T_i), we first generate the single-graph predictions where each uses different graph information. In the following, we show how to generate the prediction based on features learned from $\mathcal{G}_d$. First, the query representation and table representation are fed into two separate multi-layer perceptrons (MLP):

$$h'^{(k)}_{qd} = MLP^{(k)}_{qd}(h^{(k)}_{qd}), \qquad h'^{(k)}_{ti} = MLP^{(k)}_{ti}(h^{(k)}_{ti}) \tag{7.8}$$

Then a neural tensor network (NTN) is used to generate the prediction at k-th layer (Equation (7.9)).

$$y^{(k)}_d = h'^{\mathbf{T}(k)}_{qd} W^{(k)}_2 h'^{(k)}_{ti} + W^{(k)}_3 \begin{bmatrix} h'^{(k)}_{qd} \\ h'^{(k)}_{ti} \end{bmatrix} + b^{(k)}_1 \tag{7.9}$$

We use a linear layer to combine the predictions from all layers.

$$y_d = [y^{(0)}_d; ...; y^{(l)}_d]W_4 + b_2 \tag{7.10}$$

The prediction y_c based on the embeddings of $\mathcal{G}_c$ is similar to the steps in Equations (7.8-7.10). Another linear layer is used to combine the predictions from different graphs:

$$y_g = [y_d; y_c]W_5 + b_3 \tag{7.11}$$

Our method can be easily extended to have more than two graphs. For example, we can build separate graphs for each context field and combine predictions for each graph. For the purpose of explanation, we construct only one graph for all the context fields.

We can also predict the relevance score between q and T_i only based on text information:

$$y_t = \mathit{TextEncoder}(q, text_i) \tag{7.12}$$

where *Text-Encoder*$(\cdot)$ can be any previous table retrieval method which treats tables as text documents and $text_i$ is the text representation of a table(e.g. the concatenation of terms in T_i and its context fields). We obtain the final relevance score

y by combining the graph-based and text-based predictions with a linear transformation:

$$y = [y_t; y_g]W_6 + b_4 \tag{7.13}$$

7.2.6 Model Training

To learn the parameters, we optimize the model with pointwise mean square loss as in [299, 57]:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 + \beta \cdot ||\Theta||^2 \tag{7.14}$$

where y_i is the prediction from our model and $\hat{y_i}$ is the ground truth for i-th training sample. Θ denotes all trainable parameters and β controls the L2 normalization which can affect overfitting.

7.3 Experiments

7.3.1 Dataset

To evaluate the performance of MGNETS, we utilize the WikiTables benchmark [299] which includes 1.6M tables extracted from Wikipedia articles and each table has three context fields: page title, section title and caption. We also treat table headers as an additional context field. We evaluate the results using NDCG@5, NDCG@10, P@5 and MAP. All results are tested for statistical significance using the paired Student's t-test at 95% confidence.

7.3.2 Parameter Settings

We implement our models using Pytorch and DGL[1]. The node embedding size is fixed to 50. For all multi-layer perceptrons, we use two layers. The Adam optimizer [123] is used to optimize all models. Five-fold cross validation is used to obtain the final evaluation metrics and we use the same splits as [57] for fair comparison. All the models are trained with 100 epochs. In terms of other hyperparameters, we apply a grid search method here: learning rate is searched in $[1e^{-6}, 3e^{-6}, 1e^{-4}, 3e^{-4}, 1e^{-2}, 3e^{-2}]$, and L2 normalization coefficient is tuned in $[5e^{-6}, 5e^{-5}, 5e^{-4}, 5e^{-3}]$. We only use one layer of GIN in the multi-graph encoder.

7.3.3 Baselines Settings

To demonstrate the effectiveness of our MGNETS model, we compare to the following: **Multi-field BM25** [200], which is an unsupervised method that treats tables as multifield documents and ranks tables with combined BM25 scores from each field; **STR** [299], which proposes multiple embedding-based features and different strategies to generate ranking features from those embeddings. A random forest fits the ranking features in a point-wise manner; **Conv-KNRM** [69] where convolutional neural networks are used to learn n-gram soft matching signals between queries and documents; and, **BERT-ROW-MAX** [57], the previous state-of-the-art method using BERT as the backbone. It selects the most important rows with max salience selector and concatenates them with context fields as BERT input. Due to limited computational resources, we use the BERT-base-cased[2] instead of BERT-large-cased as in the original paper to initialize the BERT component. We use Conv-KNRM or BERT-ROW-MAX as the text encoder in MGNETS, named as MGNETS-Conv and MGNETS-BERT respectively.

[1]`https://github.com/dmlc/dgl`

[2]`https://huggingface.co/bert-base-cased`

Table 7.1: Performance comparison with baselines. The superscript † denotes statistically significant improvements over all other methods.

Model	MAP	P@5	NDCG@5	NDCG@10
Multi-field BM25	0.4596	0.3273	0.4365	0.5049
STR	0.5711	0.3927	0.5762	0.6048
ConvKNRM	0.5561	0.3800	0.5556	0.5901
MGNETS-Conv	0.5912 (+6.3%)	0.3907 (+2.8%)	0.5910 (+6.4%)	0.6168 (+4.5%)
BERT-Row-Max	0.6146	0.4080	0.6167	0.6322
MGNETS-BERT	**0.6339** (+3.1%)	**0.4180**†(+2.5%)	**0.6373**†(+3.3%)	**0.6490**†(+2.7%)

7.3.4 Overall Performance

We start by comparing the performance of MGNETS with all other baselines, as reported in Table 7.1. We can observe that our proposed **MGNETS-BERT** achieves the best performance across all evaluation metrics. As a strong interaction-based neural IR model, ConvKNRM underperforms the feature-based STR model, which indicates that the table retrieval task should not be treated as a traditional document retrieval task. Combining both text-based predictions and graph-based predictions, MGNETS-BERT outperforms BERT-ROW-MAX by $2.5 - 3.3\%$ and MGNETS-Conv outperforms ConvKNRM by $2.8 - 6.4\%$. The results verify the effectiveness of our designed framework where the multi-graph information can benefit methods where only text information is used. However, we do not claim our method to be the new state-of-the-art method, since our goal is to study whether the learned graph features can provide additional signals. It is possible that combining other designed features can further improve the performance such as in Chapter 6, which is beyond the scope of this chapter.

7.3.5 Ablation Study

To evaluate the effectiveness of several key components in MGNETS, we performed ablation studies as shown in Table 7.2. The 2nd line in Table 7.2 shows the result when the text encoder was removed. Encouragingly, MGNETS still performs better

Table 7.2: Ablation study of our framework.

Model	MAP	P@5	NDCG@5	NDCG@10
MGNETS-BERT	0.6339	0.4180	0.6373	0.6490
MGNETS (graph only)	**0.5812**	**0.3913**	**0.5823**	**0.6072**
GNETS ($\mathcal{G}$)	0.5753	**0.3913**	0.5748	0.6065
GNETS ($\mathcal{G}_c$)	0.5634	0.3820	0.5657	0.6003
GNETS ($\mathcal{G}_d$)	0.5701	0.3860	0.5678	0.5975
BERT-ROW-MAX (table)	0.5859	0.3933	0.5784	0.6061
ConvKNRM (table)	0.5403	0.3787	0.5382	0.5746

than all baselines in Table 7.1 except BERT-ROW-MAX. Without ConvKNRM as the text encoder, the performance of MGNETS-Conv does not decrease too much compared with MGNETS (graph only).

We also compare the performance of the model when using a single graph instead of multiple graphs. GNETS is the model that only operates on one graph and outputs a single graph-based prediction. GNETS ($\mathcal{G}_d$) achieves better results than GNETS ($\mathcal{G}_c$) on all metrics except NDCG@10. This indicates that the features in data tables could be more effective than features in context fields. We can see that even with all types of edges, GNETS ($\mathcal{G}$) underforms MGNETS, which suggests it is more difficult for graph neural networks to extract features from one single heterogeneous graph than from separate graphs. One possible reason is that edges constructed from different fields may have different semantic meanings and constructing a single heterogeneous graph could result in an ambiguous semantic space.

We further show the results of baselines (last 2 lines in Table 7.2) when only using data tables as input (i.e., no context fields). The NDCG@5 scores of BERT-ROW-MAX and Conv-KNRM decrease by 6.2% and 3.7% respectively, while GNETS ($\mathcal{G}_d$) has similar performance with GNETS ($\mathcal{G}$). It verifies our observation that the text-based methods are less effective to extract features from flattened tables where structure information is underutilized. The results also indicate that our proposed methods are more robust when context information is missing.

7.4 Summary

In this chapter, we propose a graph-based solution MGNETS to address the task of table retrieval. We first study how to model the table corpus as one or more heterogeneous graphs and then utilize graph neural networks to learn the representations of queries and tables from complex structures. Experimental results demonstrate the features learned from multiple graphs can improve the text-based neural IR models.

7.5 Bibliographic Notes

In this section, we review related work in graph-based methods for information retrieval. Early methods such as PageRank [193] and HITS [125] model link structure among web pages and estimate their relative importance. Jiang et al. [114] build a web-scale click graph in which a node represents a query or a document. A vector propagation algorithm on the click graph is then proposed to learn vector representations for both queries and documents. A matrix factorization method is proposed by Ma et al. [156] to learn query latent features for the task of query suggestion from two bipartite graphs, where one is user-query bipartite graph and another is query-URL bipartite graph.

More recently, embedding techniques have been used in graph-based methods. Zhang et al. [306] exploit DeepWalk [199] and LINE [247] to learn embeddings for queries and products from constructed query graph and product graph. Then the learned query and product embeddings are used as features for product search. Similar to product search, Li et al. [143] also use GNNs [124] to encode graph-based information for document retrieval. In this chapter, we propose a graph-based method for tabular dataset search where the structure information in the corpus is modeled with graph neural networks.

Chapter 8

Learning of Universal Dataset Encoders

8.1 Introduction

In previous chapters, we have introduced and proposed different methods for tabular dataset search, where given a keyword query, the goal is to return a list of tabular datasets in descending order of their relevance scores with respect to the user query. In database community, the task of finding unionable and joinable tables with a source table has been studied for many years. It can also be considered as one type of dataset search task where the query is also a dataset [265, 178, 305, 301, 264].

Current dataset search engines do not index dataset content and one reason is that the raw datasets can be in different format. However, it can be helpful to build a search engine where users can upload a dataset in specific format and retrieve relevant datasets. Here we give a motivating scenario. Bob, a social scientist, has a project analyzing public opinions about COVID-19 through social media platforms such as Twitter. To do this, Bob first has crawled a large number of tweets which include certain hashtags (e.g., #covid19). One way to analyze the opinions is to treat it as a text classification problem where labels indicate users' opinions or sentiments. Annotating all of those data is impossible for Bob and annotating

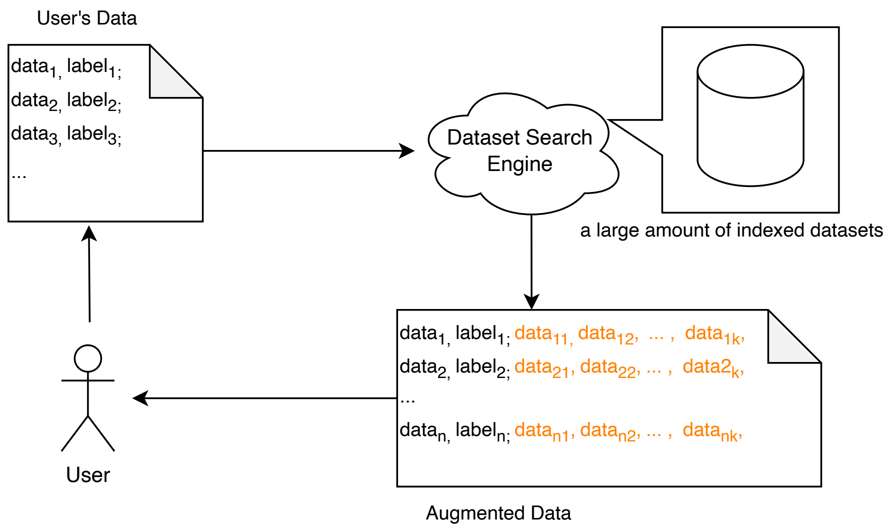

User's Data
data1, label1;
data2, label2;
data3, label3;
...
Dataset Search Engine
a large amount of indexed datasets
data1, label1; data11, data12, ... , data1k,
data2, label2; data21, data22, ... , data2k,
...
datan, labeln; datan1, datan2, ... , datank,
User
Augmented Data

Figure 8.1: A motivating example of dataset search where query is also a dataset. The data in orange is retrieved from the dataset search engine.

few samples is feasible. We can assume there is a dataset search engine which has indexed many datasets. Among those indexed datasets, some datasets may be annotated by other researchers studying similar problems. For example, Mohammad et al. [170] propose a dataset including more than 22,000 annotated tweets for the task of emotion classification. Intuitively, this dataset is somewhat relevant to Bob's dataset. First, both datasets are collected from Twitter and therefore share some characteristics (e.g., length). Second, both datasets are created for similar purposes which indicates a label in one dataset could also be meaningful in another dataset. We denote Bob's annotated dataset as $\mathcal{D}_b = \{d_1, d_2, ..., d_n\}$. A dataset search engine $\mathcal{R}$ takes d_i as input and returns top-k relevant samples (i.e., tweets) from all indexed datasets. There are two potential ways of using the data in Mohammad et al. [170] to solve Bob's project.

1. **Few Shot Learning** [240]: pretraining a model such as BERT [77] on retrieved data and then fine-tuning the model on a small set of Bob's annotated data;

2. **Weakly-supervised Learning** [165, 144, 81]: using bootstrapping strategies to label the retrieved data and then training a classifier on both datasets.

This application scenario is shown in Figure 8.1.

In this chapter, we show how to learn representations for datasets in a single format. Specifically, we introduce a method called **Universal Dataset Encoders (UDE)** that can encode a dataset into a dense representation. The learned dataset representations from UDE can can help downstream tasks such as dataset retrieval and dataset clustering.

8.2 Methods

8.2.1 Datasets are Point Sets

A point set seems to be a good intermediate representation for datasets in many modalities. Some datasets are naturally point sets. For example, a point cloud is a set of unordered 3D data points in space which represent a 3D shape or object. A trajectory is a set of ordered 2D (or 3D) points describing the movement of an object. In fact, text documents are treated as point sets by modern natural language processing techniques. Either traditional bag-of-words models [221], or more recent deep embedding-based methods [166, 77], consider a document as a set of word representations. And for other datasets, they can often be transformed into point sets. An image of size $a \times b$ can be translated into ab points in the format of (C_{ab}, y_{ab}), where C_{ab} is the coordinate of a pixel and y_{ab} is the corresponding pixel value.

Therefore, we propose to represent a dataset as a point set. Formally, each dataset can be represented as $x = \{p_1, ..., p_i, ..., p_m\}$ where p_i is a k-dimensional data point and m is the number of points inside x. Note that the size of each dataset can be different. As proposed in Figure 8.2, we propose the framework of universal dataset encoder where the similarity between a query dataset $x_q \in Q$ and a candidate dataset can be calculated so that the most similar datasets to x_q can be retrieved. In the following subsections, we introduce different Point Set Encoders that can learn features for a point set.

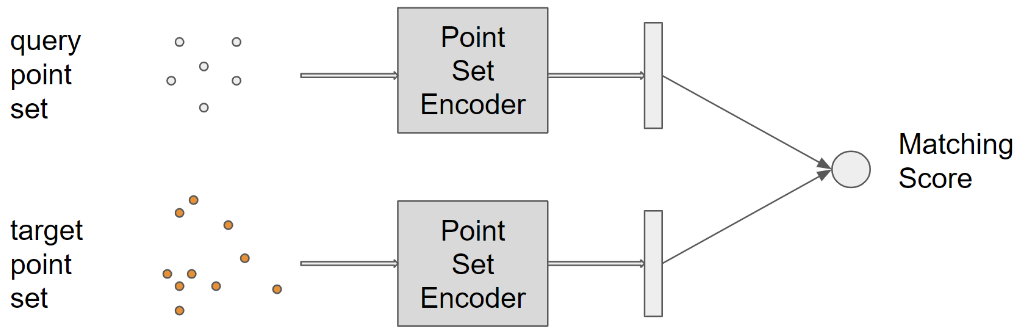


Figure 8.2: Architecture of universal dataset encoder.

8.2.2 PointNet++ Encoder

We use PointNet++ [205] as the point set encoder as it was originally applied to point cloud classification and segmentation. Here we briefly introduce the three components of PointNet++:

- **Sampling layer.** Given a point set $x = \{p_1, ..., p_i, ..., p_m\}$, iterative farthest point sampling (FPS) is used to choose a subset of points $x_s = \{p_{i_1}, p_{i_2}, ..., p_{i_n}\}$ where p_{i_j} is the most distant point from $\{p_{i_1}, p_{i_2}, ..., p_{i_{j-1}}\}$ with regard to the rest of the points. x_s can be regarded as an abstract sketch of x so that the size of x can be controlled.

- **Grouping layer.** In order to build local regions of a point set, the grouping layer groups the input points into multiple groups (clusters) using centroids from the previous sampling layer. Each group is constructed from a centroid with K nearest neighbor (kNN) search.

- **PointNet layer.** In each local region, PointNet [204] is used to extract features of each local region from the grouping layer. First, the coordinates of points in a local region are translated into a local frame relative to the centroid point: $p_i = p_i - \hat{p}$ where $\hat{p}$ is the centroid of this local region. Then the following function is used to extract local region features:

$$f(p_1, ..., p_i, ..., p_m) = \gamma(\max_{i=1,...,m}(h(p_i))$$

where γ and h are multi-layer perceptron (MLP) networks.

The above three components form one PointNet++ layer and we can stack multiple PointNet++ layers.

8.2.3 Graph Encoder

The input of the PointNet++ Encoder is the set of raw dataset points. The structure of a point set is implicitly used in the sampling layer (i.e., FPS process) and grouping layer (KNN search). An alternative way is explicitly constructing a graph from a point set so that modern graph neural networks (GNNs) can be used as graph encoders to learn point set representations.

Algorithm 1: The Ball-tree construction process of a point set.

```
   Input  : A point set $x = \{p_1, ..., p_i, ..., p_m\}$;
   Output: The root of a constructed ball tree $B$.
1  if a single point remains then
2      create a leaf $B$ containing the single point in $x$;
3      return $B$;
4  else
5      choose dimension c which has the greatest spread of points;
6      let p be the central point selected considering c;
7      let L, R be the point sets lying to the left and right of the median along
         dimension c, respectively;
8      create $B$ with two children;
9      $B$.pivot = p;
       /* recursively constructing the remaining part */
10     $B$.child1 = construct(L);
11     $B$.child2 = construct(R);
12     $B$.radius = max(distance(p,bc)), where bc is a child node of $B$;
13     return $B$;
14 end
```

Here, we first introduce the **Dataset Index Graph (DIG)**. Given a point set x, we can create its corresponding DIG G_x with a ball tree data structure which is efficient for organizing points in a multi-dimensional space. The construction of a ball tree is a top-down process that recursively splits the data points into two sets. Each split step chooses the dimension with the greatest spread of points and partitions the data by the median value along that dimension. We show the construction process of a ball tree in Algorithm 1[1].

By organizing a set of data points with such a hierarchical data structure, meaningful properties about the data distribution can be captured. A leaf node in G_x represents an original data point and a non-leaf node of G_x represents a centroid of a subset of original data points which are close to each other. Then we can use any GNN layer as the point set encoder to learn the dataset representation. In our experiments, we choose Graph Convolutional Networks (GCNs) [124] as the graph encoder.

When the size of the point set is large, the Dataset Index Graph generated from Algorithm 1 is a good data structure to organize those data points. However, if the size of the point set is small, it may be unnecessary to build a DIG following Algorithm 1. For example, a short text may only have a few words (i.e., points) while the dimension of word embeddings can be 300, which makes the dimension search step inefficient. Therefore, we propose **Sentence Index Graph (SIG)** for short text documents. First, we build a single DIG for the whole vocabulary from pre-trained word embeddings (e.g., GloVe [198]). Then we construct the SIG by visiting each point (or word) appeared in the short text on the vocabulary DIG and back-tracing the links to a parent node. Each SIG is a sub-graph of the vocabulary DIG. The benefit of using SIG is that it implicitly utilizes the corpus-level word semantics in the vocabulary DIG.

[1]The algorithm is adapted from Omohundro [188]

8.2.4 Matching of Point Sets

Given a Point Set Encoder $\mathcal{E}$ and two point sets $x_1 \in \mathcal{R}^k$ and $x_2 \in \mathcal{R}^k$, we first obtain the dataset representations:

$$r_1 = \mathcal{E}(x_1) \tag{8.1}$$

$$r_2 = \mathcal{E}(x_2) \tag{8.2}$$

Then we use a linear layer to calculate the final matching score s ($0 \leq s \leq 1$) between x_1 and x_2:

$$s = [r_1 \oplus r_2]M \tag{8.3}$$

where $\oplus$ concatenates the representation of r_1 and r_2. $M \in \mathbb{R}^{2*d\times 1}$ is a trainable variable and d (set as 50 in our experiments) is the dimension of the dataset representations. During inference, the time complexity of Equation 8.3 is $O(d)$. Distance metrics for point sets (e.g., Hausdorff distance) usually have time complexity of $O(m_1 \times m_2)$ where m_1 and m_2 are the size of the two point sets, respectively. Usually, d can be very small and the size of a point set can be very large. Though our current framework needs additional cost to obtain r_1 and r_2, dense retrieval techniques [293] can be applied in the future to pre-compute dataset representations efficiently and can be ignored during inference.

8.2.5 Training of Point Set Encoder

To learn the parameters, we optimize the model with pointwise mean squared loss:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(s_i - \hat{s_i})^2 + \beta \cdot ||\Theta||^2 \tag{8.4}$$

where $\hat{s_i}$ is the prediction from Equation 8.3 and s_i is the ground truth for i-th training sample. Θ denotes all trainable parameters and β is the L2 normalization coefficient which controls the strength of the L2 normalization to prevent overfitting.

8.3 Experiments

8.3.1 Datasets

We evaluate the performance of UDE framework on different synthesized dataset search tasks using the following datasets:

- **Porto** [172]: This is a trajectory dataset collected from 442 taxis in the city of Porto, Portugal over 19 months and contains 1.7 million trajectories. It was originally proposed for the task of predicting the destination of taxi trips based on initial partial trajectories.

- **ShapeNet** [45]: ShapeNet is point cloud dataset containing about 51,300 3D CAD models from 55 common object categories (e.g., table, car, airplane, etc.).

- **MixText**: We synthesize MixText from 17 popular text classification datasets: IMDB reviews [158], AG's News [303], DBpedia ontology classification dataset [303], Amazon reviews [303], Yelp reviews [303], BANKING77 [39] and 11 datasets from TweetEval [18].

Among the above datasets, the first two datasets are naturally datasets of point sets. Porto are ordered point sets, while ShapeNet and MixText are unordered point sets. In fact, the point set representations for text documents can be considered as one type of Bag-of-Words representations because the order information is lost. It is also one of the weaknesses of unordered point set representations.

For Porto and ShapeNet, we sample 1,000 point sets, respectively. We calculate the Hausdorff distance d_h between two point sets and use $1 - d_h$ as the ground-truth dataset similarity, which results in 1,000 $\times$ 1,000 pairs. We use 20% pairs for training and the rest for testing.

To synthesize the training set of MixText, we use the training set portion of each text collection. We sample 20 texts from each text collection, and for each text, we sample one positive text from the same text collection and one negative text from another text collection. In other words, two texts form a positive pair if they are from the same text collection, or form a negative pair if they are from different text collections. To synthesize the testing set of MixText, we use the testing set portion of each text collection. We sample 5 texts from each text collection. For each text, we increase the number of positive samples to 10 and negative samples to 500. So the evaluation for each query dataset (a text) is based on the ranking the 510 texts.

8.3.2 Results

We compare the performance of PointNet++ Encoder and our Graph Encoder in Table 8.1 with Precision@k and Recall@k. We stack two layers of GCN [124] as the graph encoder. The Adam optimizer [123] is used to optimize all models. We train each model for 30 epochs with learning rate of 1e-3 and batch size of 200.

For MixText, we also compare with sentence-BERT [216] which is a state-of-the-art method for calculating similarities between sentence pairs. We use the BERT-base-cased[2] to initialize the sentence-BERT. We train sentence-BERT with 10 epochs with batch size of 16 and maximum input length of 128. The Adam optimizer with learning rate of 1e-5 is used. We also use a linear learning rate decay schedule with warm-up of 0.1. For Porto and ShapeNet, we use DIG representations as input of graph encoders. For MixText, we use SIG representations as input of graph encoders.

Table 8.1: Results on different datasets.

Dataset	Encoder Type	P@5	P@10	P@50	R@5	R@10	R@50
Porto	Graph Encoder	**0.6769**	**0.6333**	**0.4781**	**0.0677**	**0.1267**	**0.4781**
	PointNet++	0.6398	0.5554	0.3707	0.0640	0.1111	0.3707
ShapeNet	Graph Encoder	**0.4000**	**0.6000**	**0.4800**	**0.0400**	**0.1200**	**0.4800**
	PointNet++	**0.4000**	**0.6000**	0.2800	0.0400	0.0600	0.3000
MixText	Graph Encoder	**0.4014**	**0.4316**	**0.2333**	**0.1396**	**0.3031**	**0.8228**
	PointNet++	0.3653	0.3936	0.2317	0.1330	0.2858	0.8156
	sentence-BERT	0.1906	0.1376	0.0588	0.0953	0.1376	0.2941

[2]`https://huggingface.co/bert-base-cased`

From the results, we can see that the graph encoder usually performs better than PointNet++ on different modalities. However, graph encoder shows less advantage on ShapeNet since PointNet++ is originally designed for point cloud datasets. For both encoders, the top-k precision decreases when k increases from 10 to 50 due to the difficulty of the task.

8.4 Summary

This chapter presents a novel method of learning universal dataset encoders. We argue that datasets in various formats are either point sets or can be translated into point sets. Then we can adapt Siamese networks to learn dense representations for datasets which can be used for retrieving similar datasets with a query dataset.

8.5 Bibliographic Notes

Retrieving similar datasets with an example dataset has been studied by multiple communities. In this section, we review the recent work of similar dataset search for datasets in different modalities.

Ranking a collection of documents according to their semantic similarity to a query document is an important task in information retrieval. Ginzburg et al. [96] adopt the RoBERTa [153] language model as a backbone and continue training the model with a self-supervised target where sentence pairs sampled from the same document are positive samples and sentence pairs sampled from different documents are negative samples. During inference, two input documents are decomposed into sentences and a final similarity score is reduced from a pairwise sentence-similarity matrix.

Retrieving related images from a database with an input image is also called Content-Based Image Retrieval (CBIR) in the computer vision community [133, 101]. El-Nouby et al. [84] propose to train a transformer model [256] with a Siamese architecture for image retrieval. Two images are mapped into a common latent space by the transformers. Contrastive loss combined with an entropy regularizer is used to train the resulting model.

Finding similar trajectories given a query trajectory [266, 267] is a fundamental task in smart city applications such as ridesharing [233] and traffic analysis [308, 99]. The first deep learning approach to learn representations of trajectories are proposed by Li et al. [145]. They first partition the space into cells of equal size and then map a trajectory into a sequence of discrete tokens. A seq2seq-based model [243] is used to encode a trajectory and recover its distorted counterpart. Wang et al. [273] propose the task of subtrajectory search problem which aims to return a portion of a trajectory that is the most similar to a query trajectory. A series of algorithms are developed which include both exact and approximate ones.

Depending on the characteristics of different modalities, the pre-processing steps and model structures can be very different. In this chapter, we propose our preliminary study on an intermediate representation in which the same model can be adopted to learn representations for datasets in different modalities and the representations can be used for dataset search.

Chapter 9

Conclusion and Future Work

In this chapter, we conclude this dissertation. We first summarize our research findings. Then, we discuss several future directions in dataset search.

9.1 Summary

With a growing number of datasets, dataset search has become an important task in recent years. Making datasets discoverable like Web pages is meaningful to facilitate dataset reuse and generate real value from data. In this dissertation, we first systematically introduce the various techniques that can help build a dataset search engine. Then we propose various methods for dataset augmentation and dataset search. Our detailed contributions are summarized as follow:

- We review technologies from multiple communities (i.e., data management, machine learning, semantic web and information retrieval) that can be used to build a dataset search engine. We break down the pipeline of a dataset search engine into four parts. We discuss how to extract or crawl datasets from different sources, what tasks can be done to better manage datasets and different methods of multiple types of dataset search tasks.
- An ontology designed specifically for AI-related papers and an information extraction framework is proposed. The ontology defines important classes

and properties about AI tasks. A relation classification method is adopted to extract data from research papers. We show the extracted data can be used to construct a knowledge base and support academic applications including academic dataset search.

- A feature-based approach to solve the task of schema label generation is proposed. Based on our observations, we propose a list of heuristics to extract features from a table column and treat the schema label generation as a multi-class classification task. Through schema label generation, more common (and thus understandable) schema labels can be provided to allow for broader schema matches in contexts such as dataset search and data linking. In our experiments, we find that our method often gives predictions that are synonyms or hypernyms of the original schema label.

- A schema label enhanced ranking framework for tabular dataset search is proposed. The framework has two stages: in the first stage, a schema label generator is trained to generate additional schema labels for each dataset column; in the second stage, given a user query, datasets are ranked by their original fields together with generated schema labels. Instead of using hand-curated features, we learn the latent feature representations of schema labels by a CoFactor model in which the dataset-schema label interactions and schema label-schema label interactions are captured. Our experimental results show that combining the scores for generated schema labels with the traditional ranking scores for text fields can help rank the datasets better.

- A table search method based on the deep contextualized language model BERT is proposed. We study how to encode table content considering the table structure and input length limit of BERT. We also propose an approach that incorporates features from prior work on table retrieval and jointly trains them with BERT. Our experimental results show that using the max salience selector with row items is the best strategy to construct BERT input.

- A new test collection for the task of Web table retrieval is proposed. Compared with previous datasets, WTR covers a broader range of topics and includes tables from over 61,000 different domain names. Since a Web table usually has rich context information such as the page title and surrounding paragraphs, we not only provide relevance judgments of query-table pairs, but also the relevance judgments of different table contexts with respect to a query, which are ignored by previous test collections. In our experiments, we show that the relevance of context fields can affect the training of table retrieval methods.

- A graph-based table search method is proposed. We build one or more graphs from the whole table corpus so that complex relations are captured. Then graph neural networks are used to learn the representations of queries and tables from complex structures. Experimental results demonstrate the features learned from multiple graphs can improve upon the text-based neural IR models which treat a table as a flattened document. We also find that our graph-based model is more robust when context information of a table is missing.

- A new framework to learn the representations of datasets is proposed, representing a dataset as a point set. A graph-based data structure is developed to organize point sets. In our experiments, we show that this representation learning framework can be applied to datasets in different modalities and the learned representations can be used for dataset search where the query is also a dataset.

9.2 Future Work

9.2.1 Transfer Learning of Dataset Search

It is an interesting question to study the transferability of different models and what features extracted from datasets in one domain also generalize to dataset from another domain. Wikipedia pages usually have a common structure and the extracted tabular datasets are well-formatted. However, the structure of different websites can be very different. For example, the page title of Wikipedia usually represents an entity or a very informative event. While for some websites, the page title can be very general and does not describe anything specific about the Web page. Therefore, it is possible that some features extracted from one domain do not generalize well to another domain. When we reproduce the features of LTR and STR for the WTR collection proposed in Chapter 6, we disregard some features proposed for Wikipedia specifically.

Recently, pre-training methods [77] show they have the ability to transfer knowledge learned from pre-training tasks to downstream applications. One possible direction is to propose pre-training tasks for dataset search. For example, we can mask some content in the metadata (e.g., title) and then recover it. We can also construct a knowledge graph from datasets [34] and then adopt pre-training strategies [107, 207] for graphs. Multimodal pre-training has been studied to learn cross-modal representations which can perform well on multi-modal tasks [14, 208, 268, 235, 290]. Since the content of datasets can be in different modalities and the metadata can be considered as text, multimodal pre-training can be applied to datasets to learn representations that are more robust and generalizable.

9.2.2 Multifield Scoring

The importance of metadata fields can vary depending on the source of datasets. A dataset is often associated with multiple metadata fields and can be considered as a multifield document. It has been shown that combining similarities and rankings of different sections in a document can lead to better performance for Web document retrieval [275]. Ogilvie et al. [187] present a mixture-based language model combining different document representations for known-item search in structured document collections. They find that document representations that perform poorly can be combined with other representations to improve the overall performance. Robertson et al. [223] introduce BM25F which is an extension of BM25 that combines the original term frequencies in the different fields in a weighted manner. A field relevance model is proposed by Kim and Croft [121] to incorporate relevance feedback for field weights estimation. A Bayesian networks-based model for structured documents is proposed by Piwowarski and Gallinari [202]. Kim et al. [122] propose a probabilistic model for semi-structured document retrieval. They calculate the mapping probability of each query term and use it as a weight to combine the language models estimated from each field. Svore et al. [244] develop LambdaBM25, a machine learning approach to BM25-style retrieval that learns from the input attributes of BM25 and performs better than BM25F for multifield document ranking. Zamani et al. [291] propose a neural ranking model that learns an aggregated document representation from field-level representations and then uses a matching network to produce the final relevance score. In Chapter 4, we propose an unsupervised method to score different sections of a dataset. In the future, we believe the supervised methods proposed for multifield document retrieval task can also help dataset search where metadata fields are scored with learned weights.

9.2.3 Search-Centric Applications

Dataset search can serve as an intermediate step for many downstream applications [300]. Retrieved tabular datasets can be further used as input for open domain question answering systems [195, 194]. In transfer learning, finding relevant datasets for pretraining (source domain) is critical for a model to have a good performance when fine-tuned with a small set of labelled data on a downstream task (target domain) [181, 67, 283]. Retrieved unlabelled data can also be used as weakly-supervised signals for various machine learning tasks [165, 144, 81].

In the future, dataset search may become an infrastructure-level service for AI applications where clients are different AI models. Benefiting from the large number of indexed datasets, an AI model can be trained with active learning strategies more easily [129]. A major obstacle to train a Web-scale dataset search engine is the lack of training data. The interaction between a dataset search engine and its clients can provide task-specific training signals. One possible future direction is using reinforcement learning to maximize the rewards that indicate the performance of downstream applications. For example, the evaluation score such as accuracy from a downstream question answering system can be used as a reward to train the dataset search engine. Since different downstream applications may use different evaluation metrics, multiple types of rewards can be used as training signals.

9.3 Conclusion

In this dissertation, we develop methods to tackle the problems of dataset search and augmentation. We show how data can be extracted from research articles to support scholar dataset search. We analyze the patterns in datasets and propose methods to generate table schema labels using features extracted from datasets. For unsupervised tabular dataset search, we propose to rank a dataset with generated schema labels. For supervised tabular dataset search, we propose a pre-trained language model-based method to extract powerful features and a graph-based model to learn robust features. We also propose a representation learning framework that unifies the training target of dataset representation learning for different modalities. Experimental results on various datasets show that we advance the state-of-the-art in dataset search and augmentation. Future work may consider: (1) adopting transfer learning strategies to enable dataset search from different domains; (2) designing supervised models to utilize multiple metadata fields; (3) building an infrastructure-level dataset search engine to support various downstream data-centric applications.

Bibliography

[1] Faheem Abbas, Muhammad Kamran Malik, Muhammad Umair Rashid, and Rizwan Zafar. WikiQA—a question answering system on wikipedia using freebase, dbpedia and infobox. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 185–193. IEEE, 2016.

[2] Amira Abd El-atey, Sherif El-etriby, et al. Semantic data extraction from infobox wikipedia template. *International Journal of Computer Applications*, 975:8887, 2012.

[3] Tarek Amr Abdallah and Beatriz de La Iglesia. URL-based web page classification: With n-gram language models. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*, pages 19–33. Springer, 2014.

[4] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *The VLDB Journal*, 24(4):557–581, 2015.

[5] Marco D Adelfio and Hanan Samet. Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment*, 6(6):421–432, 2013.

[6] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. Towards a hybrid imputation approach using web tables. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 21–30. IEEE, 2015.

[7] Uchenna Akujuobi and Xiangliang Zhang. Delve: a dataset-driven scholarly search and analysis system. *ACM SIGKDD Explorations Newsletter*, 19(2):36–46, 2017.

[8] Marcel Altendeitering, ISST Fraunhofer, and Tobias Guggenberger. Designing data quality tools: findings from an action design research project at boehringer ingelheim. In *Proceedings of the 29th European Conference on Information Systems*, page 17, 2021.

[9] Yael Amsterdamer and Tova Milo. Foundations of crowd data sourcing. *ACM SIGMOD Record*, 43(4):5–14, 2015.

[10] Tom Anderson, ZH Andrews, JS Fitzgerald, Brian Randell, Hugh Glaser, and IC Millard. The resist resilience knowledge base. *School of Computing Science Technical Report Series*, 2007.

[11] Hiba Arnaout and Shady Elbassuoni. Effective searching of rdf knowledge graphs. *Journal of Web Semantics*, 48:66–84, 2018.

[12] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[13] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In *The semantic web (ISWC)*, pages 722–735. Springer, 2007.

[14] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. See, hear, and read: Deep aligned representations. *arXiv preprint arXiv:1706.00932*, 2017.

[15] Paul Azunre, Craig Corcoran, David Sullivan, Garrett Honke, Rebecca Ruppel, Sandeep Verma, and Jonathon Morgan. Abstractive tabular dataset summarization via knowledge base semantic embeddings. *arXiv preprint arXiv:1804.01503*, 2018.

[16] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 384–392, 2019.

[17] Marko Balabanovic and Yoav Shoham. Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, 1997.

[18] Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa-Anke, and Leonardo Neves. TweetEval:Unified Benchmark and Comparative Evaluation for Tweet Classification. In *Proceedings of Findings of EMNLP*, 2020.

[19] Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the 16th international conference on World Wide Web*, pages 441–450, 2007.

[20] Mohamed Ben Ellefi, Zohra Bellahsene, John G Breslin, Elena Demidova, Stefan Dietze, Julian Szymański, and Konstantin Todorov. RDF dataset profiling–a survey of features, methods, vocabularies and applications. *Semantic Web*, 9(5):677–705, 2018.

[21] Mohamed Ben Ellefi, Zohra Bellahsene, Stefan Dietze, and Konstantin Todorov. Beyond established knowledge graphs-recommending web datasets for data linking. In *Int'l Conf. on Web Engineering*, pages 262–279. Springer, 2016.

[22] Mohamed Ben Ellefi, Zohra Bellahsene, Stefan Dietze, and Konstantin Todorov. Dataset recommendation for data linking: An intensional approach. In *The Semantic Web. Latest Advances and New Domains*, pages 36–51. Springer, 2016.

[23] Omar Benjelloun, Shiyu Chen, and Natasha Noy. Google dataset search by the numbers. In *International Semantic Web Conference*, pages 667–682. Springer, 2020.

[24] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *Advanced Information Systems Engineering*, pages 452–466. Springer, 2002.

[25] Tim Berners-Lee. `https://www.w3.org/DesignIssues/LinkedData.html`.

[26] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Methods for exploring and mining tables on Wikipedia. In *Proceedings of the ACM SIGKDD workshop on Interactive Data Exploration and Analytics*, pages 18–26, 2013.

[27] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. TabEL: entity linking in web tables. In *Proc. Int'l Semantic Web Conf. (ISWC)*, pages 425–441, 2015.

[28] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[29] Sara Bonfitto, Luca Cappelletti, Fabrizio Trovato, Giorgio Valentini, and Marco Mesiti. Semi-automatic column type inference for csv table understanding. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 535–549. Springer, 2021.

[30] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[31] Charalampos Bratsas, Lazaros Ioannidis1 Dimitris Kontokostas, Sören Auer, Christian Bizer, Sebastian Hellmann, and Ioannis Antoniou. DBpedia internationalization-a graphical tool for i18n infobox-to-ontology mappings. In *International Semantic Web Conference Demo (ISWC2011 Demo)*, 2011.

[32] Katrin Braunschweig, Maik Thiele, and Wolfgang Lehner. From web tables to concepts: A semantic normalization approach. In *International Conference on Conceptual Modeling*, pages 247–260. Springer, 2015.

[33] Dan Brickley, Matthew Burgess, and Natasha Noy. `https://www.kaggle.com/datasets/googleai/dataset-search-metadata-for-datasets`.

[34] Dan Brickley, Matthew Burgess, and Natasha Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, pages 1365–1375, 2019.

[35] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009.

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

[37] Michael J Cafarella, Alon Y Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the relational web. In *WebDB*, pages 1–6, 2008.

[38] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1335–1349, 2020.

[39] Iñigo Casanueva, Tadas Temcinas, Daniela Gerz, Matthew Henderson, and Ivan Vulic. Efficient intent detection with dual sentence encoders. In *Proceedings of the 2nd Workshop on NLP for ConvAI - ACL 2020*, mar 2020. Data available at https://github.com/PolyAI-LDN/task-specific-datasets.

[40] Silvana Castano and Valeria De Antonellis. Global viewing of heterogeneous data sources. *IEEE Trans. on Knowledge and Data Eng.*, 13(2):277–297, 2001.

[41] Sonia Castelo, Rémi Rampin, Aécio Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. Auctus: a dataset search engine for data discovery and augmentation. *Proceedings of the VLDB Endowment*, 14(12):2791–2794, 2021.

[42] Taha Ceritli, Christopher KI Williams, and James Geddes. ptype: probabilistic type inference. *Data Mining and Knowledge Discovery*, 34(3):870–904, 2020.

[43] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer networks*, 31(11-16):1623–1640, 1999.

[44] Zhangming Chan, Xiuying Chen, Yongliang Wang, Juntao Li, Zhiqiang Zhang, Kun Gai, Dongyan Zhao, and Rui Yan. Stick to the facts: Learning towards a fidelity-oriented e-commerce product description generation. In *Proceedings of the 2019 Conference on EMNLP and the 9th IJCNLP*, pages 4958–4967, 2019.

[45] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[46] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit Dhillon. X-BERT: extreme multi-label text classification with using bidirectional encoder representations from transformers. In *Proceedings of NeurIPS Science Meets Engineering of Deep Learning Workshop*, 2019.

[47] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. Dataset search: a survey. *The VLDB Journal*, 29(1):251–272, 2020.

[48] Hsin-Hsi Chen, Shih-Chung Tsai, and Jin-He Tsai. Mining tables from large scale html texts. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*, 2000.

[49] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. Colnet: Embedding the semantics of web tables for column type prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 29–36, 2019.

[50] Minmin Chen. Efficient vector representation for documents through corruption. *5th International Conference on Learning Representations*, 2017.

[51] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. Open question answering over tables and text. In *International Conference on Learning Representations*, 2021.

[52] Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. HybridQA: A dataset of multi-hop question answering over tabular and textual data. *Findings of the Association for Computational Linguistics (EMNLP)*, pages 1026–1036, 2020.

[53] Zhe Chen and Michael Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, pages 1–8, 2013.

[54] Zhiyu Chen, Haiyan Jia, Jeff Heflin, and Brian D Davison. Generating schema labels through dataset content analysis. In *Companion Proceedings of the The Web Conference 2018*, pages 1515–1522, 2018.

[55] Zhiyu Chen, Haiyan Jia, Jeff Heflin, and Brian D Davison. Leveraging schema labels to enhance dataset search. In *European Conference on Information Retrieval*, pages 267–280. Springer, 2020.

[56] Zhiyu Chen, Mohamed Trabelsi, Brian D. Davison, and Jeff Heflin. Towards knowledge acquisition of metadata on ai progress. In *CEUR workshop proceedings*, volume 2721, 2020.

[57] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D Davison. Table search using a deep contextualized language model. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 589–598, 2020.

[58] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Dawei Yin, and Brian D. Davison. MGNETS: Multi-graph neural networks for table search. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2945–2949, 2021.

[59] Zhiyu Chen, Shuo Zhang, and Brian D. Davison. WTR: A test collection for web table retrieval. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2514–2520, 2021.

[60] Christina Christodoulakis, Eric B Munson, Moshe Gabel, Angela Demke Brown, and Renée J Miller. Pytheas: pattern-based table discovery in csv files. *Proceedings of the VLDB Endowment*, 13(12):2075–2089, 2020.

[61] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does bert look at? an analysis of bert's attention. In *BlackBoxNLP@ACL*, 2019.

[62] GeoBlacklight community. `https://geoblacklight.org/about`.

[63] Kaggle community. `https://www.kaggle.com/about/team`.

[64] Eli Cortez, Philip A Bernstein, Yeye He, and Lev Novik. Annotating database schemas to help enterprise search. *Proceedings of the VLDB Endowment*, 8(12):1936–1939, 2015.

[65] Eric Crestan and Patrick Pantel. Web-scale table census and classification. In *Proceedings 4th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 545–554. ACM, 2011.

[66] Silviu Cucerzan and Eugene Agichtein. Factoid question answering over unstructured and structured web content. In *TREC*, volume 72, page 90, 2005.

[67] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118, 2018.

[68] Zhuyun Dai and Jamie Callan. Deeper text understanding for ir with contextual neural language modeling. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 985–988, July 2019.

[69] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 126–134, 2018.

[70] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

[71] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 817–828, 2012.

[72] data.world. `http://data.world`.

[73] Hélio Rodrigues de Oliveira, Alberto Trindade Tavares, and Bernadette Farias Lóscio. Feedback-based data set recommendation for building linked data applications. In *Proc. 8th Int'l Conf. on Semantic Systems*, I-SEMANTICS '12, pages 49–55. ACM, 2012.

[74] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: table understanding through representation learning. *Proceedings of the VLDB Endowment*, 14(3):307–319, 2020.

[75] Harsh Desai, Pratik Kayal, and Mayank Singh. TabLeX: a benchmark dataset for structure and content information extraction from scientific tables. In *International Conference on Document Analysis and Recognition*, pages 554–569. Springer, 2021.

[76] Danilo Dessì, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, Enrico Motta, and Harald Sack. AI-KG: an automatically generated knowledge graph of artificial intelligence. In *International Semantic Web Conference*, pages 127–143. Springer, 2020.

[77] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[78] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C Lee Giles, and Marco Gori. Focused crawling using context graphs. In *VLDB*, 2000.

[79] Li Ding, Dominic DiFranzo, Alvaro Graves, James R Michaelis, Xian Li, Deborah L McGuinness, and Jim Hendler. Data-gov Wiki: Towards linking government data. In *2010 AAAI Spring Symposium Series*, 2010.

[80] Hong-Hai Do and Erhard Rahm. COMA-a system for flexible combination of schema matching approaches. In *Proc. 28th Int'l Conf. on Very Large Databases*, pages 610–621. Elsevier, 2002.

[81] Thomas Dopierre, Christophe Gravier, Julien Subercaze, and Wilfried Logerais. Few-shot pseudo-labeling for intent detection. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4993–5003, 2020.

[82] Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. Building the resden web table corpus: A classification approach. In *IEEE/ACM 2nd Int'l Symp. on Big Data Computing (BDC)*, pages 41–50, 2015.

[83] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *International Semantic Web Conference*, pages 260–277. Springer, 2017.

[84] Alaaeldin El-Nouby, Natalia Neverova, Ivan Laptev, and Hervé Jégou. Training vision transformers for image retrieval. *arXiv preprint arXiv:2102.05644*, 2021.

[85] Shady Elbassuoni and Roi Blanco. Keyword search over rdf graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 237–242, 2011.

[86] Jing Fang, Prasenjit Mitra, Zhi Tang, and C. Lee Giles. Table header detection and classification. In *Proc. 26th AAAI Conf. on Artificial Intelligence*, AAAI'12, pages 599–605. AAAI Press, 2012.

[87] Open Knowledge Foundation. `http://ckan.org`.

[88] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[89] Kata Gábor, Davide Buscaldi, Anne-Kathrin Schumann, Behrang Qasem-iZadeh, Haifa Zargayouna, and Thierry Charnois. SemEval-2018 task 7: Semantic relation extraction and classification in scientific papers. In *Proceedings of the 12th International Workshop on Semantic Evaluation*, pages 679–688, 2018.

[90] Kata Gábor, Haïfa Zargayouna, Davide Buscaldi, Isabelle Tellier, and Thierry Charnois. Semantic annotation of the ACL anthology corpus for the automatic analysis of scientific literature. In *Proceedings of the LREC 2016 Conference*, May 2016.

[91] Kyle Yingkai Gao and Jamie Callan. Scientific table search using keyword queries. *arXiv preprint arXiv:1707.03423*, 2017.

[92] Hector Garcia-Molina, Manas Joglekar, Adam Marcus, Aditya Parameswaran, and Vasilis Verroios. Challenges in data crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 28(4):901–911, 2016.

[93] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th international conference on World Wide Web*, pages 71–80, 2007.

[94] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[95] Saheli Ghosh, Tin Vu, Mehrad Amin Eskandari, and Ahmed Eldawy. UCR-STAR: The ucr spatio-temporal active repository. *SIGSPATIAL Special*, 11(2):34–40, 2019.

[96] Dvir Ginzburg, Itzik Malkiel, Oren Barkan, Avi Caciularu, and Noam Koenigstein. Self-supervised document similarity ranking via contextualized language

models and hierarchical inference. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3088–3098, Online, August 2021. Association for Computational Linguistics.

[97] Koraljka Golub and Anders Ardö. Importance of HTML structural elements and metadata in automated subject classification. In *International Conference on Theory and Practice of Digital Libraries*, pages 368–378. Springer, 2005.

[98] Carlos A Gomez-Uribe and Neil Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13, 2016.

[99] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 922–929, 2019.

[100] Maryam Habibi, Johannes Starlinger, and Ulf Leser. TabSim: A siamese neural network for accurate estimation of table similarity. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 930–937. IEEE, 2020.

[101] Ibtihaal M Hameed, Sadiq H Abdulhussain, and Basheera M Mahmmod. Content-based image retrieval: A review of recent trends. *Cogent Engineering*, 8(1):1927469, 2021.

[102] Shuo Han, Lei Zou, Jeffery Xu Yu, and Dongyan Zhao. Keyword search on RDF graphs-a query graph assembly approach. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 227–236, 2017.

[103] Faegheh Hasibi, Krisztian Balog, Darío 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*, pages 1289–1292, 2017.

[104] Yufang Hou, Charles Jochim, Martin Gleize, Francesca Bonin, and Debasis Ganguly. Identification of tasks, datasets, evaluation metrics, and numeric scores for scientific leaderboards construction. In *57th ACL*, pages 5203–5213, July 2019.

[105] Yufang Hou, Charles Jochim, Martin Gleize, Francesca Bonin, and Debasis Ganguly. TDMSci: A specialized corpus for scientific literature entity tagging of tasks datasets and metrics. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 707–714, Online, April 2021. Association for Computational Linguistics.

[106] Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. Retrieve, read, rerank: Towards end-to-end multi-document reading comprehension. In *Proc. 57th An. Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 2285–2295, 2019.

[107] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. GPT-GNN: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2020.

[108] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[109] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1500–1508, 2019.

[110] Matthew Hurst. Layout and language: Challenges for table understanding on the web. In *Proc. Int'l Workshop on Web Document Analysis*, pages 27–30, 2001.

[111] Yusra Ibrahim, Mirek Riedewald, and Gerhard Weikum. Making sense of entities and quantities in web tables. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1703–1712, 2016.

[112] Sarthak Jain, Madeleine van Zuylen, Hannaneh Hajishirzi, and Iz Beltagy. SciREX: A challenge dataset for document-level information extraction. In *Proc. 58th Annual Meeting of the Association for Computational Linguistics*, pages 7506–7516, Online, July 2020.

[113] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[114] Shan Jiang, Yuening Hu, Changsung Kang, Tim Daly Jr, Dawei Yin, Yi Chang, and Chengxiang Zhai. Learning query and document relevance from a web-scale click graph. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 185–194, 2016.

[115] Wittawat Jitkrittum, Choochart Haruechaiyasak, and Thanaruk Theeramunkong. Qast: Question answering system for Thai Wikipedia. In *Proceedings of the 2009 Workshop on Knowledge and Reasoning for Answering Questions (KRAQ 2009)*, pages 11–14, 2009.

[116] Min-Yen Kan and Hoang Oanh Nguyen Thi. Fast webpage classification using url features. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 325–326, 2005.

[117] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*, pages 3363–3372, 2011.

[118] Marcin Kardas, Piotr Czapla, Pontus Stenetorp, Sebastian Ruder, Sebastian Riedel, Ross Taylor, and Robert Stojnic. Axcell: Automatic extraction of results from machine learning papers. *arXiv preprint arXiv:2004.14356*, 2020.

[119] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics.

[120] Md Abu Kausar, VS Dhaka, and Sanjeev Kumar Singh. Web crawler: a review. *International Journal of Computer Applications*, 63(2), 2013.

[121] Jin Young Kim and W Bruce Croft. A field relevance model for structured document retrieval. In *Proc. European Conf. on Info. Retrieval*, pages 97–108. Springer, 2012.

[122] Jinyoung Kim, Xiaobing Xue, and W Bruce Croft. A probabilistic retrieval model for semistructured data. In *Proc. European Conference on Information Retrieval (ECIR)*, pages 228–239. Springer, 2009.

[123] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[124] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[125] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[126] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[127] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 468–479. IEEE, 2021.

[128] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on EMNLP and the 9th IJCNLP (EMNLP-IJCNLP)*, pages 4364–4373, 2019.

[129] Anita Krishnakumar. Active learning literature survey. *Tech. rep., Technical reports, University of California, Santa Cruz.*, 42, 2007.

[130] Yamuna Krishnamurthy, Kien Pham, Aecio Santos, and Juliana Freire. Interactive web content exploration for domain discovery. *Interactive Data Exploration and Analytics (IDEA) Workshop at Knowledge Discovery and Data Mining (KDD)*, 2016.

[131] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[132] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *TACL*, 7:453–466, 2019.

[133] Afshan Latif, Aqsa Rasheed, Umer Sajid, Jameel Ahmed, Nouman Ali, Naeem Iqbal Ratyal, Bushra Zafar, Saadat Hanif Dar, Muhammad Sajid, and Tehmina Khalil. Content-based image retrieval and feature extraction: a comprehensive review. *Mathematical Problems in Engineering*, 2019, 2019.

[134] Larissa R Lautert, Marcelo M Scheidt, and Carina F Dorneles. Web table taxonomy and formalization. *ACM SIGMOD Record*, 42(3):28–33, 2013.

[135] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[136] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy, July 2019. Association for Computational Linguistics.

[137] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[138] Oliver Lehmberg and Christian Bizer. Web table column categorisation and profiling. In *Proceedings of the 19th International Workshop on Web and Databases*, pages 1–7, 2016.

[139] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 75–76, 2016.

[140] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. The Mannheim search join engine. *Journal of Web Semantics*, 35:159–166, 2015.

[141] Luiz André P Paes Leme, Giseli Rabello Lopes, Bernardo Pereira Nunes, Marco Antonio Casanova, and Stefan Dietze. Identifying candidate datasets for data interlinking. In *Int'l Conf. on Web Engineering*, pages 354–366. Springer, 2013.

[142] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[143] Xiangsheng Li, Maarten de Rijke, Yiqun Liu, Jiaxin Mao, Weizhi Ma, Min Zhang, and Shaoping Ma. Learning better representations for neural information retrieval with graph information. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 795–804, 2020.

[144] Ximing Li and Bo Yang. A pseudo label based dataless naive bayes algorithm for text classification with seed words. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1908–1917, 2018.

[145] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th international conference on data engineering (ICDE)*, pages 617–628. IEEE, 2018.

[146] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM conference on recommender systems*, pages 59–66. ACM, 2016.

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

[148] Jimmy Lin and Boris Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *Proc. 12th Int'l Conf. on Information and Knowledge Management*, CIKM '03, pages 116–123. ACM, 2003.

[149] Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. Denoising distantly supervised open-domain question answering. In *Proc. 56th Annual Meeting of*

the Assoc. for Computational Linguistics (Vol. 1: Long Papers), pages 1736–1745, 2018.

[150] Xien Liu, Xinxin You, Xiao Zhang, Ji Wu, and Ping Lv. Tensor graph convolutional networks for text classification. In *AAAI*, pages 8409–8416, 2020.

[151] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. TableRank: A ranking algorithm for table search and retrieval. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 317. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[152] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. TableSeer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 91–100, 2007.

[153] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT pretraining Approach. arXiv preprint arXiv:1907.11692, 2019.

[154] Giseli Rabello Lopes, Luiz André P Paes Leme, Bernardo Pereira Nunes, and Marco A Casanova. RecLAK: Analysis and recommendation of interlinking datasets. In *LAK Workshops*, 2014.

[155] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proc. Empirical Methods in Natural Language Processing*, pages 3219–3232, October-November 2018.

[156] Hao Ma, Haixuan Yang, Irwin King, and Michael R Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 709–718, 2008.

[157] Xiaofei Ma, Peng Xu, Zhiguo Wang, Ramesh Nallapati, and Bing Xiang. Universal text representation from BERT: An empirical study. *arXiv preprint arXiv:1910.07973*, 2019.

[158] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[159] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. CEDR: Contextualized embeddings for document ranking. In *Proc. 42nd Int'l ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1101–1104, 2019.

[160] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proc. 27th Int'l Conf. on Very Large Data Bases*, VLDB '01, pages 49–58. Morgan Kaufmann, 2001.

[161] D-Lib Magazine. The landscape of research data repositories in 2015: A re3data analysis. *D-Lib Magazine*, 23(3/4), 2017.

[162] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. Yago3: A knowledge base from multilingual Wikipedias. In *7th biennial conference on innovative data systems research*. CIDR Conference, 2014.

[163] Yosi Mass, Haggai Roitman, Shai Erera, Or Rivlin, Bar Weiner, and David Konopnicki. A study of BERT for non-factoid question-answering under passage length constraints. *arXiv preprint arXiv:1908.06780*, 2019.

[164] Filippo Menczer, Gautam Pant, and Padmini Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Transactions on Internet Technology (TOIT)*, 4(4):378–419, 2004.

[165] Yu Meng, Jiaming Shen, Chao Zhang, and Jiawei Han. Weakly-supervised neural text classification. In *proceedings of the 27th ACM International Conference on information and knowledge management*, pages 983–992, 2018.

[166] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[167] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[168] Renée J Miller. Open data integration. *Proceedings of the VLDB Endowment*, 11(12):2130–2139, 2018.

[169] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[170] Saif Mohammad, Felipe Bravo-Marquez, Mohammad Salameh, and Svetlana Kiritchenko. SemEval-2018 task 1: Affect in tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 1–17, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[171] Alvaro Morales, Varot Premtoon, Cordelia Avery, Sue Felshin, and Boris Katz. Learning to answer questions from Wikipedia infoboxes. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1930–1935, 2016.

[172] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, 2013.

[173] Heiko Müller, Sonia Castelo, Munaf Qazi, and Juliana Freire. From papers to practice: the openclean open-source data cleaning library. *Proceedings of the VLDB Endowment*, 14(12):2763–2766, 2021.

[174] Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic message passing for generating linked data from tables. In *The Semantic Web – ISWC 2013*, pages 363–378. Springer, 2013.

[175] Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi. T2LD: Interpreting and representing tables as linked data. In *9th Int'l Semantic Web Conf. (ISWC)*, page 25, 2010.

[176] Marc Najork. Web crawler architecture., 2009.

[177] Fatemeh Nargesian, Ken Q Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J Miller. Organizing data lakes for navigation. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1939–1950, 2020.

[178] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. Table union search on open data. *Proceedings of the VLDB Endowment*, 11(7):813–825, 2018.

[179] Zara Nasar, Syed Waqar Jaffry, and Muhammad Kamran Malik. Information extraction from scientific articles: a survey. *Scientometrics*, 117(3):1931–1990, 2018.

[180] Alfredo Nazabal, Pablo M Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020.

[181] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V Le, and Ruoming Pang. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, 2018.

[182] Andriy Nikolov and Mathieu d'Aquin. Identifying relevant sources for data linking using a semantic web index. In *WWW2011 Workshop: Linked Data on the Web (LDOW 2011)*, 2011.

[183] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. What should i link to? identifying relevant sources and classes for data linking. In *The Semantic Web*, pages 284–299. Springer, 2012.

[184] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*, 2020.

[185] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-stage document ranking with BERT. *arXiv preprint arXiv:1910.14424*, 2019.

[186] Blaž Novak. A survey of focused web crawling algorithms. *Proceedings of SIKDD*, 5558:55–58, 2004.

[187] Paul Ogilvie and Jamie Callan. Combining document representations for known-item search. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 143–150. ACM, 2003.

[188] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[189] Fernando Ortega, José-Luis Sánchez, Jesúa Bobadilla, and Abraham Gutiérrez. Improving collaborative filtering-based recommender systems results using pareto dominance. *Information Sciences*, 239:50–61, 2013.

[190] Masayo Ota, Heiko Müller, Juliana Freire, and Divesh Srivastava. Data-driven domain discovery for structured datasets. *Proceedings of the VLDB Endowment*, 13(7):953–967, 2020.

[191] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q Pu, and Renée J Miller. RONIN: data lake exploration. *Proceedings of the VLDB Endowment*, 14(12):2863–2866, 2021.

[192] Harshith Padigela, Hamed Zamani, and W Bruce Croft. Investigating the successes and failures of bert for passage re-ranking. *arXiv preprint arXiv:1905.01758*, 2019.

[193] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[194] Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. End-to-end table question answering via retrieval-augmented generation. *arXiv preprint arXiv:2203.16714*, 2022.

[195] Feifei Pan, Mustafa Canim, Michael R. Glass, A. Gliozzo, and Peter Fox. CLTR: An end-to-end, transformer-based system for cell-level table retrieval and table question answering. *ArXiv*, abs/2106.04441, 2021.

[196] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[197] Boya Peng, Yejin Huh, Xiao Ling, and Michele Banko. Improving knowledge base construction from robust infobox extraction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 138–148, 2019.

[198] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[199] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[200] Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. *Proceedings of the VLDB Endowment*, 5:908–919, 2012.

[201] David Pinto, Michael Branstein, Ryan Coleman, W Bruce Croft, Matthew King, Wei Li, and Xing Wei. Quasm: a system for question answering using semi-structured data. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 46–55, 2002.

[202] Benjamin Piwowarski and Patrick Gallinari. A machine learning model for information retrieval with structured documents. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 425–438. Springer, 2003.

[203] Mahima Pushkarna and Andrew Zaldivar. Data cards: Purposeful and transparent documentation for responsible AI. *ACM SIGKDD Explorations Newsletter*, 2022.

[204] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[205] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 30, 2017.

[206] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of BERT in ranking. *arXiv preprint arXiv:1904.07531*, 2019.

[207] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020.

[208] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021.

[209] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, Dec 2001.

[210] R Rajalakshmi and Chandrabose Aravindan. An effective and discriminative feature learning for URL based web page classification. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1374–1379. IEEE, 2018.

[211] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[212] J-Y Ramel, Michel Crucianu, Nicole Vincent, and Claudie Faure. Detection, extraction and representation of tables. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 374–378. IEEE, 2003.

[213] Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Anish Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 50–61. IEEE, 2013.

[214] Lev Ratinov and Ehud Gudes. Abbreviation expansion in schema matching and web integration. In *Proc. IEEE/WIC/ACM Int'l Conf. on Web Intelligence*, pages 485–489, 2004.

[215] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, 2009.

[216] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[217] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, aug 2017.

[218] Petar Ristoski and Heiko Paulheim. RDF2Vec: RDF graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

[219] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. Matching HTML tables to DBpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, pages 1–6, 2015.

[220] Dominique Ritze, Oliver Lehmberg, Yaser Oulabi, and Christian Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proc. 25th Int'l Conf. on World Wide Web*, WWW '16, pages 251–261, 2016.

[221] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 2004.

[222] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[223] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. In *Proc. 13th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 42–49, 2004.

[224] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[225] Negar Rostamzadeh, Diana Mincu, Subhrajit Roy, Andrew Smart, Lauren Wilcox, Mahima Pushkarna, Jessica Schrouff, Razvan Amironesei, Nyalleng Moorosi, and Katherine Heller. Healthsheet: Development of a transparency artifact for health datasets, 2022.

[226] Wataru Sakata, Tomohide Shibata, Ribeka Tanaka, and Sadao Kurohashi. FAQ retrieval using query-question similarity and BERT-based query-answer relevance. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 1113–1116, 2019.

[227] Angelo A Salatino, Thiviyan Thanapalasingam, Andrea Mannocci, Francesco Osborne, and Enrico Motta. The computer science ontology: a large-scale taxonomy of research areas. In *International Semantic Web Conference*, pages 187–205. Springer, 2018.

[228] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[229] Amany M Sarhan, Ghada M Hamissa, and Heba E Elbehiry. Feature selection algorithms based on HTML tags importance. In *2015 Tenth International Conference on Computer Engineering & Systems (ICCES)*, pages 185–190. IEEE, 2015.

[230] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[231] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[232] Yoones A Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. Knowledge base augmentation using tabular data. In *LDOW*, 2014.

[233] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S Jensen, Kai Zheng, and Panos Kalnis. Parallel trajectory similarity joins in spatial networks. *The VLDB Journal*, 27(3):395–420, 2018.

[234] Roee Shraga, Haggai Roitman, Guy Feigenblat, and Mustafa Cannim. Web table retrieval using multimodal deep learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1399–1408, 2020.

[235] Amanpreet Singh, Ronghang Hu, Vedanuj Goswami, Guillaume Couairon, Wojciech Galuba, Marcus Rohrbach, and Douwe Kiela. FLAVA: A foundational language and vision alignment model. In *CVPR*, 2022.

[236] Serena Sorrentino, Sonia Bergamaschi, and Maciej Gawinecki. NORMS: an automatic tool to perform schema label normalization. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1344–1347. IEEE, 2011.

[237] Serena Sorrentino, Sonia Bergamaschi, Maciej Gawinecki, and Laura Po. Schema normalization for improving schema matching. In *Conceptual Modeling - ER 2009*, pages 280–293. Springer, 2009.

[238] Heinrich Stamerjohanns, Michael Kohlhase, Deyan Ginev, Catalin David, and Bruce Miller. Transforming large collections of scientific publications to XML. *Mathematics in Computer Science*, 3(3):299–307, 2010.

[239] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. *arXiv preprint arXiv:2104.01785*, 2021.

[240] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? In *China national conference on Chinese computational linguistics*, pages 194–206. Springer, 2019.

[241] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. Table cell search for question answering. In *Proceedings of the 25th International Conference on World Wide Web*, pages 771–782, 2016.

[242] Yibo Sun, Zhao Yan, Duyu Tang, Nan Duan, and Bing Qin. Content-based table retrieval for web queries. *Neurocomputing*, 349:183–189, 2019.

[243] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[244] Krysta M Svore and Christopher JC Burges. A machine learning approach for improved BM25 retrieval. In *Proc. 18th ACM Conf. on Information and Knowledge Management (CIKM)*, pages 1811–1814, 2009.

[245] Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. Exploiting a web of semantic data for interpreting tables. In *Proc. 2nd Web Science Conf.*, volume 5, 2010.

[246] Ki Hyun Tae, Yuji Roh, Young Hun Oh, Hyunsu Kim, and Steven Euijong Whang. Data cleaning for accurate, fair, and robust models: A big data-AI integration approach. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, pages 1–4, 2019.

[247] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[248] Mohamed Trabelsi, Jin Cao, and Jeff Heflin. SeLaB: Semantic labeling with bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[249] Mohamed Trabelsi, Zhiyu Chen, Brian D Davison, and Jeff Heflin. A hybrid deep model for learning to rank data tables. In *IEEE International Conference on Big Data (Big Data)*, pages 979–986. IEEE, 2020.

[250] Mohamed Trabelsi, Zhiyu Chen, Brian D. Davison, and Jeff Heflin. Relational graph embeddings for table retrieval. In *IEEE International Conference on Big Data (Big Data)*, pages 3005–3014. IEEE, 2020.

[251] Mohamed Trabelsi, Zhiyu Chen, Brian D Davison, and Jeff Heflin. Neural ranking models for document retrieval. *Information Retrieval Journal*, 24(6):400–444, 2021.

[252] Mohamed Trabelsi, Zhiyu Chen, Shuo Zhang, Brian D Davison, and Jeff Heflin. StruBERT: Structure-aware bert for table search and matching. In *Proceedings of the ACM Web Conference 2022*, pages 442–451, 2022.

[253] Mohamed Trabelsi, Brian D Davison, and Jeff Heflin. Improved table retrieval using multiple context embeddings for attributes. In *IEEE International Conference on Big Data (Big Data)*, pages 1238–1244. IEEE, 2019.

[254] Barbara Ubaldi. Open government data: Towards empirical analysis of open government data initiatives. *OECD Working Papers on Public Governance*, page 0_1, 2013.

[255] Isabel Valera and Zoubin Ghahramani. Automatic discovery of the statistical types of variables in a dataset. In *Proc. 34th Int'l Conf. on Machine Learning*,

volume 70 of *Proceedings of Machine Learning Research*, pages 3521–3529. PMLR, 06–11 Aug 2017.

[256] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[257] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations*, 2018.

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

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

[260] A Gural Vural, B Barla Cambazoglu, and Pinar Karagoz. Sentiment-focused web crawling. *ACM Transactions on the Web (TWEB)*, 8(4):1–21, 2014.

[261] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR*, 2019.

[262] Jiannan Wang, Guoliang Li, and Jianhua Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *2011 IEEE 27th International Conference on Data Engineering*, pages 458–469. IEEE, 2011.

[263] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. Understanding tables on the web. In *Conceptual Modeling*, pages 141–155. Springer, 2012.

[264] Sheng Wang, Zhifeng Bao, J Shane Culpepper, and Gao Cong. A survey on trajectory data management, analytics, and learning. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021.

[265] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, Mark Sanderson, and Xiaolin Qin. Answering top-k exemplar trajectory queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 597–608. IEEE, 2017.

[266] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. Torch: A search engine for trajectory data. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 535–544, 2018.

[267] Wei Wang, Feng Xia, Hansong Nie, Zhikui Chen, Zhiguo Gong, Xiangjie Kong, and Wei Wei. Vehicle trajectory clustering based on dynamic representation learning of internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 22(6):3567–3576, 2020.

[268] Wenhui Wang, Hangbo Bao, Li Dong, and Furu Wei. VLMo: Unified vision-language pre-training with mixture-of-modality-experts. *arXiv preprint arXiv:2111.02358*, 2021.

[269] Yalin Wang and Jianying Hu. Detecting tables in HTML documents. In *International Workshop on Document Analysis Systems*, pages 249–260. Springer, 2002.

[270] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th international conference on World Wide Web*, pages 242–250, 2002.

[271] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *Proc. 11th Int'l Conf. on World Wide Web*, WWW '02, pages 242–250. ACM, 2002.

[272] Zhen Wang, Jiachen Liu, Xinyan Xiao, Yajuan Lyu, and Tian Wu. Joint training of candidate extraction and answer selection for reading comprehension. In *Proceedings of the 56th Annual Meeting of the ACL*, pages 1715–1724, 2018.

[273] Zheng Wang, Cheng Long, Gao Cong, and Yiding Liu. Efficient and effective similar subtrajectory search with deep reinforcement learning. *Proc. VLDB Endow.*, 13(12):2312–2325, jul 2020.

[274] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In *EMNLP-IJCNLP 2019*, pages 5877–5881, Hong Kong, China, November 2019. ACL.

[275] Ross Wilkinson. Effective retrieval of structured documents. In *Proc. ACM SIGIR Int'l Conf. on Research and Dev. in Information Retrieval*, pages 311–317. Springer, 1994.

[276] Sam Wiseman, Stuart Shieber, and Alexander Rush. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[277] Tianxing Wu, Shengjia Yan, Zhixin Piao, Liang Xu, Ruiming Wang, and Guilin Qi. Entity linking in web tables with multiple linked knowledge bases. In *Joint International Semantic Technology Conference*, pages 239–253. Springer, 2016.

[278] Xueqing Wu, Jiacheng Zhang, and Hang Li. Text-to-table: A new way of information extraction. *arXiv preprint arXiv:2109.02707*, 2021.

[279] Yang Xiao, Jinlan Fu, Weizhe Yuan, Vijay Viswanathan, Zhoumianze Liu, Yixin Liu, Graham Neubig, and Pengfei Liu. DataLab: A platform for data analysis and intervention, 2022.

[280] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*, 2021.

[281] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[282] Wenyuan Xue, Baosheng Yu, Wen Wang, Dacheng Tao, and Qingyong Li. TGRNet: A table graph reconstruction network for table structure recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1295–1304, 2021.

[283] Xi Yan, David Acuna, and Sanja Fidler. Neural data server: A large-scale search engine for transfer learning data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3893–3902, 2020.

[284] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-end open-domain question answering with BERTserini. In *NAACL-HLT (Demonstrations)*, pages 72–77, 2019.

[285] Wei Yang, Haotian Zhang, and Jimmy Lin. Simple applications of BERT for ad hoc document retrieval. *arXiv preprint arXiv:1903.10972*, 2019.

[286] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377, 2019.

[287] Yang Yi, Zhiyu Chen, Jeff Heflin, and Brian D Davison. Recognizing quantity names for tabular data. In *ProfS/KG4IR/Data: Search@ SIGIR*, 2018.

[288] Minoru Yoshida, Kentaro Torisawa, and Jun'ichi Tsujii. A method to integrate tables of the World Wide Web. In *Proceedings of the International Workshop on Web Document Analysis (WDA 2001)*, pages 31–34, 2001.

[289] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, 41(3):793–819, 2014.

[290] Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, et al. Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*, 2021.

[291] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. Neural ranking models with multiple document fields. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 700–708, 2018.

[292] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing dense retrieval model training with hard negatives. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1503–1512, 2021.

[293] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. Learning to retrieve: How to train a dense retrieval model effectively and efficiently. *arXiv preprint arXiv:2010.10469*, 2020.

[294] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. Sato: Contextual semantic type detection in tables. *Proc. VLDB Endow.*, 13(12):1835–1848, 2020.

[295] Haoxiang Zhang, Aécio Santos, and Juliana Freire. DSDD: Domain-specific dataset discovery on the web. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2527–2536, 2021.

[296] Li Zhang, Shuo Zhang, and Krisztian Balog. Table2Vec: Neural word and entity embeddings for table population and retrieval. In *Proc. 42nd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 1029–1032, 2019.

[297] Shuo Zhang and Krisztian Balog. Design patterns for fusion-based object retrieval. In Joemon M Jose, Claudia Hauff, Ismail Sengor Altıngovde, Dawei Song, Dyaa Albakour, Stuart Watt, and John Tait, editors, *Advances in Information Retrieval*, pages 684–690, 2017.

[298] 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, New York, NY, USA, 2017. ACM.

[299] Shuo Zhang and Krisztian Balog. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 World Wide Web Conference*, pages 1553–1562, 2018.

[300] Shuo Zhang and Krisztian Balog. Web table extraction, retrieval, and augmentation: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(2):1–35, 2020.

[301] Shuo Zhang and Krisztian Balog. Semantic table retrieval using keyword and table queries. *ACM Transactions on the Web (TWEB)*, pages 1–33, 2021.

[302] Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. Novel entity discovery from web tables. In *Proceedings of The Web Conference*, pages 1298–1308, 2020.

[303] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[304] Xiaolu Zhang, Yueguo Chen, Jinchuan Chen, Xiaoyong Du, and Lei Zou. Mapping entity-attribute web tables to web-scale knowledge bases. In *Database Systems for Advanced Applications*, pages 108–122. Springer, 2013.

[305] Yi Zhang and Zachary G Ives. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1951–1966, 2020.

[306] Yuan Zhang, Dong Wang, and Yan Zhang. Neural IR meets graph embedding: A ranking model for product search. In *The World Wide Web Conference*, pages 2390–2400, 2019.

[307] Ziqi Zhang. Towards efficient and effective semantic table interpretation. In *International Semantic Web Conference*, pages 487–502. Springer, 2014.

[308] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):1–41, 2015.

[309] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. JOSIE: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 847–864, 2019.

[310] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. LSH ensemble: Internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, aug 2016.

Vita

1993	Born in Shimen, Hunan Province, China.
2011	Graduated from the Shimen No.1 High School, Hunan Province, China.
2011 - 2015	B.S. in Computer Science and Technology, Nanjing University of Aeronautics and Astronautics.
2015 - 2022	Graduate study in Department of Computer Science and Engineering, Lehigh University.
2016	Summer internship at Bloomberg, NJ, USA.
2019	Summer internship at Zhuiyi Technology, Shenzhen, China.
2021	Summer internship at Amazon, WA, US (remote)

LIST OF PUBLICATIONS

1. **Z. Chen**, H. Jia, J. Heflin, and B.D. Davison. Generating Schema Labels through Dataset Content Analysis. In *International Workshop on Profiling and Searching Data on the Web*, 2018.

2. Y. Yi, **Z. Chen**, J. Heflin and B.D. Davison. Recognizing Quantity Names for Tabular Data. In *Joint Proceedings of the First International Workshop on Professional Search; the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding; and the International Workshop on Data Search*, 2018.

3. **Z. Chen**, H. Jia, J. Heflin, and B.D. Davison. Leveraging Schema Labels to Enhance Dataset Search. In *Proceedings of the 42nd European Conference on Information Retrieval (ECIR)*, 2020.

4. H. Ye, **Z. Chen**, D. Wang and B. D. Davison. Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Cluster for Extreme Multi-label Text Classification. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

5. **Z. Chen**, M. Trabelsi, J. Heflin, Y. Xu and B.D. Davison. Table Search Using a Deep Contextualized Language Model. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2020.

6. **Z. Chen***, M. Trabelsi*, J. Heflin and B.D. Davison. Towards Knowledge Acquisition of Metadata on AI Progress. In *Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice, co-located with the 19th International Semantic Web Conference (ISWC)*, 2020.

7. M. Trabelsi*, **Z. Chen***, J. Heflin and B.D. Davison. Relational Graph Embeddings for Table Retrieval. In *the Seventh International Workshop on High Performance Big Graph Data Management, Analysis, and Mining (BigGraphs)*, 2020.

8. M. Trabelsi, **Z. Chen**, J. Heflin and B.D. Davison. A Hybrid Deep Model for Learning to Rank Data Tables. In *Proceedings of the 2020 IEEE International Conference on Big Data (BigData)*, 2020.

9. **Z. Chen**, S. Zhang and B.D. Davison. WTR: A Test Collection for Web Table Retrieval. In *Proceedings of the 44rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2021.

10. M. Trabelsi, **Z. Chen**, J. Heflin and B.D. Davison. Neural Ranking Models for Document Retrieval. In *Information Retrieval Journal*, 2021.

11. **Z. Chen**, M. Trabelsi, J. Heflin, D. Yin and B.D. Davison. MGNETS: Multi-Graph Neural Networks for Table Search. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM)*, 2021.

12. M. Trabelsi, **Z. Chen**, S. Zhang, J. Heflin, and B.D. Davison. StruBERT: Structure-aware BERT for Table Search and Matching. In *Proceedings of the 31st Web Conference*, 2022.