Leveraging Dataset Content in Neural Models for Search and Curation

by

Mohamed Trabelsi

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.

Mohamed Trabelsi
Leveraging Dataset Content in Neural Models for Search and Curation

Date

Prof. Jeff Heflin, Dissertation Director, Chair

Accepted Date

Committee Members:

Prof. Brian D. Davison

Prof. Lifang He

Prof. Haiyan Jia

Acknowledgements

I would like to thank my advisor, Prof. Jeff Heflin for his guidance, assistance, and critical insights during my PhD at Lehigh University. We have worked together on various projects since Fall 2018, and Prof. Heflin helped me conduct solid research starting from brainstorming of research ideas to writing research papers. In addition, Prof. Heflin was very supportive in difficult times, and achieving this milestone was only possible because of his constant encouragement and confidence.

I also would like to thank everyone in the Dataset Search group. I really learned from discussing research ideas within the Dataset Search group. I would especially like to thank Prof. Brian D. Davison for providing me the opportunity to contribute to the dataset search project. Having successful research papers was only possible because of the valuable feedback and critical insights of Prof. Davison. I was always impressed by the importance of the questions and suggestions of Prof. Davison that helped me to have a solid work. I also would like to thank Prof. Haiyan Jia for the great discussions within the Dataset Search group about many aspects of the search task, that helped me to have a better understanding about how individuals search for datasets and how to assess the quality of datasets.

I would like to thank my thesis committee members: Prof. Haiyan Jia, Prof. Lifang He, and Prof. Brian D. Davison for their feedback and suggestions about the dissertation.

I also want to give a special thanks to Dr. Jin Cao for her guidance during my internships. We managed to have research papers during summer internships that helped me to have a solid dissertation. I always enjoyed research discussions and casual conversations with Dr. Jin Cao. Thank you for your support and for the opportunity to apply research ideas to solve multiple challenging real-world problems.

Parts of this work were based on funding from NSF under the Grant No. IIS-1816325.

Now, it is the part where I should thank my family, and it is very difficult to write only a couple of sentences for that. My mother, Mounira, this dissertation is definitely for you because you always believed in me, and you sacrificed everything for me. My brother, Yessin, this accomplishment is also for you, and I am very grateful to having you in my life. I also would like to thank my sisters Samiha and Neila for their support and unconditional love. Finally, I would like to also dedicate this dissertation to the soul of my dear father Abdelaziz and my dear sister Sabeh.

Contents

List of Tables

List of Figures

Abstract

With the substantial progress of deep learning techniques, learning from data and automating repetitive tasks are increasingly explored in multiple fields ranging from decision making to scientific experiments. Datasets represent a particular form of data that is explored in the data science field where the objective of learning from datasets is to extract useful knowledge for developing data-driven solutions. The process of generating useful knowledge from datasets starts with a first step that consists of retrieving relevant datasets to a user's query, and a second step that consists of cleaning datasets to improve the quality of data for downstream tasks. Multiple methods for dataset search are based on matching the user's query against the metadata of datasets, and thus ignore data values – an important source of information in datasets. We distinguish two groups of limitations in previous dataset retrieval methods. The first group is related to embedding limitations where previous methods rely on pretrained embeddings from large text corpus, and ignore the characteristics of datasets. The existing embeddings are trained on large text corpus so that they do not take into account the structure and co-occurrence information in datasets. The second group of limitations is related to the deep learning architectures where previous methods in dataset search are based on neural ranking models that are proposed for document retrieval, so that datasets are linearized and considered as simple unstructured documents. Therefore, existing ranking models ignore the specific textual and structural information of datasets. After extracting relevant datasets to a user's query, a data curation step is necessary to improve data quality in order to extract useful knowledge from the retrieved datasets for downstream analytic tasks. Our objective is to develop efficient and effective dataset search and curation methods that allow users to search across all datasets on the Internet, and improve the quality of the extracted datasets for downstream tasks.

In this dissertation, we focus on three data curation tasks. The first task is semantic labeling which consists of generating a schema label for each column from a set of labels. Existing methods generate schema labels solely on the basis of their content or data values, and thus ignore the contextual information of each column when predicting schema labels. The second task is dataset similarity that can be used to cluster datasets for better organization of the dataset collections, and to make recommendations for other datasets that might be relevant. Understanding the connection between the textual and structural information is an important yet neglected aspect in table similarity as previous methods treat each source of information independently. The third task is entity matching which consists of determining whether two records refer to the same real-world entity. Despite the effort in the past years to improve the performance in entity matching, the existing methods still require a huge amount of labeled data in each domain during the training phase. These methods treat each domain individually, and capture the specific signals for each dataset in entity matching, and this leads to overfitting on just one dataset. The knowledge that is learned from one dataset is not explored to better understand the entity matching task in order to make predictions on the unseen datasets with fewer labeled samples.

In this dissertation, we propose multiple methods to overcome the limitations of existing approaches in dataset search and retrieval. The first research direction focuses on the embedding part for datasets to overcome the limitations of the traditional context-independent embedding for tokens. The second research direction focuses on the neural architecture design to incorporate the structural information of datasets into the ranking process, and overcome the limitations of document retrieval based models that are applied for dataset search. In addition, we propose multiple methods for the three data curation tasks with the ultimate objective is to automate the process of extracting useful knowledge from datasets.

Chapter 1

Introduction

1.1 Overview

Vast amounts of information that are related to scientific, cultural, and political topics, are stored in datasets. Many users have questions that can be resolved from this data, but these questions may go unanswered due to ignorance regarding the presence of the data, ignorance regarding where to look for the data, and inability to formulate queries using the domain-specific vocabulary of the data's creators. Given that many datasets are in tabular form, for the rest of the dissertation, we use data tables and datasets interchangeably. A data table has multiple rows and columns. Each column can be seen as a variable described by a schema label in order to distinguish between the variables. The decentralized aspect of the Web allows individuals and organizations to publish data; the fast progress in information retrieval has enabled access to a growing multitude of data sources. In order to extract useful knowledge from the retrieved datasets for downstream tasks, a data curation step is necessary to improve data quality. Some of the data tables are pre-processed, for example, those found in repositories such as UCI machine learning repository[1], kaggle[2], and OpenML [245]. Many governments also share repositories of tabular open data such as data.gov[3]. Some others require extraction such as those HTML tables that are embedded in web pages, spreadsheet files and PDFs. For pre-processed tables, data providers

[1]http://archive.ics.uci.edu/ml/index.php
[2]https://www.kaggle.com/
[3]https://www.data.gov/

describe their data tables using metadata, and header's names that semantically describe column values. However, many data tables do not follow metadata standards and naming standards for schema labels resulting in less informative column labels [112, 105, 33, 166]. For example, the date of birth of a person is saved as *Date of Birth* in some data tables, and *DOB* in others. On the other hand, the extracted tabular datasets can have missing or wrong header names, as a result of automatic table extraction. So, the quality of such data sources is often quite poor. The valuable knowledge, that is contained in data tables, has been explored in multiple tasks including augmenting tables [10, 17, 53, 135], extracting knowledge from tables [164], table retrieval or query answering [17, 18, 168, 38], table-based question answering [31, 225], table-to-text generation [143, 32], column relations [154, 140, 54, 20], column type annotation [275, 236, 101], entity linking [9, 295], and table type classification [48, 180].

In data table search and retrieval, tables can be considered as documents, so that document retrieval methods can be applied to table retrieval [17, 18]. The community has recognized that classical bag-of-words models are no longer state-of-the-art in part because they are not effective in capturing fine-grained contextual structures for information retrieval, and the same is true for table retrieval. Supervised learning, based on features from tables, queries, and query-table pairs [18, 10], has resulted in the best performing table retrieval systems. Building on this, Zhang and Balog [293] introduced semantic features to embed queries and tables into a semantic space, and then train a supervised model using the semantic and traditional features. However, there are major drawbacks of prior approaches for ad hoc table retrieval. First, they are based on pretrained embeddings, and they ignore contextual information within tables in the ranking step. Second, they are based on hand-crafted features, and that limits the ability to capture multiple levels of similarity between the query and table. Third, understanding the connection between the textual and structural information is an important yet neglected aspect in table retrieval as previous methods treat each source of information independently. Fourth, previous methods assume an equal contribution of each query token to the final relevance score when ranking web tables against a given query.

1.2 Objectives

Our objective is to develop efficient and effective dataset search and curation methods that allow users to search across all datasets on the Internet, and improve the quality of the retrieved datasets for downstream tasks. Dataset search and curation is a challenging big data problem due to both the volume (billions of tables) and variety (often, the schemas of tables are very different). Both dataset search and curation tasks are related, and improving the performance of one task can have a positive effect on the other task. For example, we can have a scenario where a table retrieval algorithm can be used as a core component in other tasks such as table extension [266] and table mining [209] that can result in better quality data table collections. On the other hand, we can have data curation methods that can be applied offline to datasets in order to improve the performance of dataset search and retrieval. Multiple data curation techniques can be applied to datasets to improve their quality. In this dissertation, we focus on three data curation tasks. The first task is semantic labeling which consists of generating a schema label for each column from a set of labels. In addition to data cleaning, schema labels of datasets are used in multiple tasks such as data discovery [22, 23], schema matching [197, 282] and data preparation and analysis [198]. The second task is dataset similarity that can be used to cluster datasets for better organization of the dataset collections. The third task is entity matching [4] which consists of determining whether two records refer to the same real-world entity. Despite the effort in the past years to improve the performance in entity matching, the existing methods still require a huge amount of labeled data in each domain during the training phase.

1.3 Observations

The limitations of previous table retrieval approaches can be categorized into two groups. The first group is related to **embedding limitations** where previous approaches rely on pretrained embeddings such as word embedding and graph embedding. These embeddings are trained on large text corpus so that they do not take into account the structure and co-occurrence information in data tables. The second group of limitations is related to the **neural architecture limitations** where previous methods are based on neural ranking models that are proposed for document retrieval. Designing new ranking models for data tables is inspired by classic information retrieval where multiple methods have been proposed to incorporate the internal organization of a given document into indexing and retrieval steps. Considering the structure of a document when designing retrieval models can usually improve retrieval results [260]. It has been shown that combining similarities and rankings of multiple sections can improve retrieval performance [260].

To overcome the limitations stated in the first group, we have proposed to learn a new model, called MCON [241], for word embeddings of attribute tokens that is used to predict the contextual information of tables in the ranking phase. MCON is a new model that is built for word embeddings of the tokens of table attributes using contextual information of every table, where we examined multiple formulations for contexts used to create embeddings. In addition to table retrieval, MCON can be used as a preprocessing or postprocessing tool for the attributes of data tables given that MCON provides a fixed length feature vector for the tokens of attributes. To overcome the limitations stated in the second group, we have proposed a new neural ranking model, called deep semantic and relevance matching model (DSRMM) [237], that incorporates both semantic and relevance [80] matching signals for data table search and retrieval. There are two classes of neural architectures for ad hoc retrieval. Semantic similarity architectures treat the query and target as equals, and try to match them. Query relevance architectures exploit characteristics of the ad hoc retrieval task such as exact matching and non-uniform contribution of query tokens to the final relevance score. DSRMM is a hybrid model that combines both concepts into one architecture that incorporates both semantic and relevance matching signals. In addition, DSRMM includes summary vectors about the contents of the table, both in terms of values in each column and values in selected rows. The summary vectors compress each row and each column into a fixed length feature vector using word embedding of data values.

In terms of embedding techniques, we observed three disadvantages from applying MCON either to data table retrieval or semantic labeling. First, similar to traditional word embedding techniques, such as word2vec [158] and Glove [185], MCON provides a static context-independent representation for each word. In other words, although MCON is originally trained using the contextual information of dataset collections, after the unsupervised training phase, MCON assigns a fixed embedding for each token, and a static representation for each token can negatively affects the results of table retrieval and semantic labeling. Second, MCON is mainly based on the co-occurrence information which is captured from adapting the skip-gram model [158] for tables using only textual information. In addition to the co-occurrence signal, other signals such as the semantic and lexical information, can be incorporated to improve the embedding of tokens. Inspired by document retrieval where multiple sources of information, such as text, entities, are combined to improve retrieval results, multiple external sources can be incorporated to produce embeddings for data tables and queries in multiple spaces. Third, MCON only provides embedding for the attribute's tokens. Although the attribute's embedding can be enough for semantic labeling, retrieving data tables is based on all fields that are available such as metadata and data values. To overcome the limitations of MCON, we have noticed that tables can be represented as a knowledge graph that captures dataset-specific and dataset-agnostic knowledge for both the textual and structural information of data tables. We denote this graph-based method by MultiEM-RGCN [238], where we obtain multiple types of embeddings for each token in the data table.

In terms of neural architecture techniques, although DSRMM has achieved significant improvement in data table retrieval results by including both semantic and relevance matching signals with row- and column-based summary vectors, we can cite two disadvantages for DSRMM. First, the summary vectors are computed independently of the context of data table and query using traditional pretrained word embedding. As in MCON, a context-independent representation for rows and columns in DSRMM is unable to capture the relationship between the structured form of data table, and the textual form of both metadata and queries. Second, the summary vectors are computed using the mean pooling operation, so that the data values are treated equally in summary vectors. Depending on the context of the data table defined by both the metadata and the user's query, each data value should have different contributions in both rows and columns. To overcome the limitations of DSRMM, we propose to capture the dependencies between rows and columns using a structure-aware BERT model called StruBERT [240], that fuses the structural and textual information of a data table to produce four context-aware features: two fine-grained structure- and context-aware representations for rows and columns, and two coarse-grained representations for row- and column-guided [CLS] embedding.

After finding relevant datasets to a user's query, the next step consists of dataset curation. For semantic labeling, existing methods generate schema labels solely on the basis of their content or data values, and thus ignore the contextual information of each column when predicting schema labels. We propose a new *context-aware* semantic labeling method, called SeLaB [236], that incorporates both data values and column's context in order to infer schema labels. For datasets similarity, existing methods decouple the structural information from the textual information, and StruBERT [240] can be used to fuse both sources of information and identify similar datasets. For entity matching, existing methods treat each domain individually, and capture the specific signals for each dataset, and this leads to overfitting on just one dataset. The knowledge that is learned from one dataset is not explored to better understand the entity matching task in order to make predictions on the unseen datasets with fewer labeled samples. We propose a new domain adaptation-based method, called DAME [242], that transfers the task knowledge from multiple source domains to a target domain.

1.4 Contributions

We summarize our contributions in this section. We propose multiple unsupervised and supervised methods for table retrieval, and data curation:

- We propose a new model for word embeddings of the tokens of table attributes, called MCON [241], using contextual information of every table. We demonstrate the usefulness of an attribute's collection of values (the data tokens) in creating a meaningful semantic representation of the attribute. We predict the context of tables using the trained contextual model, and we use a mixed ranking model that incorporates the metadata of a table and the additional contexts in order to calculate the retrieval score. MCON code is available on Github [4].

- We propose a new knowledge graph that incorporates multiple matching signals and external resources to learn embeddings for large collections of data tables. Our proposed graph, denoted by MutiEm-RGCN [238], includes both dataset-dependent and dataset-agnostic knowledge from table corpus. External semantic and lexical resources are used for edges and nodes leading to an heterogeneous graph. Multiple types of embeddings are learned simultaneously

[4]https://github.com/medtray/IEEEBigData2019-TablesEmbeddings

from our proposed knowledge graph using graph neural networks (GNN) with the link prediction pre-training task. Unlike MCON that provides an embedding only for the tokens of attributes, our proposed graph provides multiple embeddings for each token in all the fields. The new graph embeddings are incorporated into a learning to rank (LTR) architecture that combines multiple embeddings from our heterogeneous graph to solve the table retrieval task. MutiEm-RGCN code is available on Github [5].

- We propose a new semantic similarity matching model, called DSRMM [237], that is able to capture multiple levels of semantic signals between the query and table. Our representation of the table includes summary vectors about the contents of the table, both in terms of values in each column and values in selected rows. We demonstrate the usefulness of query relevance-specific components for the table retrieval task. Using kernel pooling, we learn a feature vector based on the probability distribution of the similarity of each document token to each query token, and we learn the contribution of each token to the final relevance score using a Term Gating Network. Each of these components lead to improvement on retrieval tasks without leading to a large increase in the number of parameters of the model. DSRMM code is available on Github [6].

- We propose a new structure-aware BERT model, called StruBERT [240], that fuses the structural and textual information of a data table to produce four context-aware features: two fine-grained structure- and context-aware representations for rows and columns, and two coarse-grained representations for row- and column-guided [CLS] embedding. We propose a new ranking model, called miniBERT, that operates directly on the embedding-level sequences formed from StruBERT features to solve three table-related downstream tasks: keyword- and content-based table retrieval, and table similarity. StruBERT code is available on Github [7].

- We propose a new context-aware semantic labeling approach, called SeLaB [236], that is used for data curation. Our new formulation of semantic labeling

[5]https://github.com/medtray/MultiEm-RGCN
[6]https://github.com/medtray/IEEEBigData2020-DSRMM-Table-Retrieval
[7]https://github.com/medtray/StruBERT

is based on the structured prediction setting in which the input to our model is a data table with missing headers, and we sequentially generate schema labels for each data table. We incorporate data values and predicted contexts using BERT, which is trained end-to-end for feature extraction and label prediction. This reduces human effort in semantic labeling. SeLaB code is available on Github [8].

- We propose a new domain adaptation-based method for entity matching denoted by DAME [242]. Our new formulation of entity matching is based on the mixture of experts where we transfer learning from multiple source domains to a target domain. We study the zero-shot learning case on the target domain and demonstrate that our method learns the entity matching task and transfers the task knowledge to the target domain. We extensively study fine-tuning our model on the target dataset from multiple domains, and demonstrate that our model generalizes better than state-of-the-art methods for most of the datasets. DAME code is available on Github [9].

1.5 Organization

The rest of the dissertation is organized as follows:

- In Chapter 2, we introduce the deep learning terminologies and techniques that are related to our proposed methods.
- In Chapter 3, we discuss the literature of table search, table similarity, semantic labeling, and entity matching.
- In Chapter4, we introduce the datasets that are used in our experiments to compare our proposed methods against baselines for table search, table similarity, semantic labeling, and entity matching.
- In Chapter 5, we present our unsupervised word embedding for tables that is based on Skipgram model, and used to predict additional contexts for table search.

[8]https://github.com/medtray/SeLaB
[9]https://github.com/medtray/DAME

- In Chapter 6, we introduce a two-phased graph-based method that is used for table retrieval, where the first phase consists of knowledge graph construction and embedding learning, and the second phase consists of incorporating the graph embeddings into a new learning-to-rank model.
- In Chapter 7, we present our neural architecture for learning to rank data tables which is a hybrid model with semantic and relevance matching components.
- In Chapter 8, we present our structure-aware BERT model for table search and matching, where we fuse the textual and structural information of data tables to solve three downstream tasks: keyword- and content-based table retrieval, and table similarity.
- In Chapter 9, we summarize our contributions in dataset search, discuss the trade-off between the effectiveness and efficiency of dataset search models, and introduce the dataset curation.
- In Chapter 10, we present our context-aware semantic labeling method that incorporates both data values and column's context to infer schema labels.
- In Chapter 11, we introduce our domain adaptation-based method that transfers knowledge from multiple source domains to a target domain in order to solve the entity matching task.
- In Chapter 12, we summarize our contributions in dataset search and curation, and we discuss future research directions about developing an end-to-end system for dataset search and curation.

Chapter 2

Related Machine Learning Techniques and Concepts

In this chapter, we introduce the deep learning terminologies and techniques that are related to our proposed methods in dataset search and curation.

2.1 N-gram Language Models

A general model for information retrieval is the N-gram Language Model (LM), which assigns probabilities to a sequence of words. Given a query, a unigram LM is used to retrieve documents by calculating the probability that each document would generate the query [45, 192, 283]. For a query Q, that is composed of query terms $q_1, q_2, \ldots, q_{|Q|}$, where $|Q|$ is the length of Q, the probability of Q given a document D is given by:

$$p(Q|\theta_d) = \prod_{i=1}^{|Q|} p(q_i|\theta_d)$$

where θ_d is the LM estimated for document D and $p(q_i|\theta_d)$ is estimated using a unigram LM. In order to avoid assigning zero probability for unseen words in a document, $p(q_i|\theta_d)$ can be estimated using a linear combination of probabilities from unigram LM and Dirichlet prior smoothing [283, 284]. For a given word w, and a collection of documents C, $p(w|\theta_d)$ is given by:

$$p(w|\theta_d) = \alpha_1 \; p(w|D) + \alpha_2 \; p(w|C) \tag{2.1}$$

with,

$$p(w|D) = \frac{count(w,D)}{|D|} \text{ and, } p(w|C) = \frac{count(w,C)}{|C|}$$

α_1 and α_2 are estimated using Dirichlet prior smoothing:

$$\alpha_1 = \frac{|D|}{|D| + \mu} \text{ and, } \alpha_2 = \frac{\mu}{|D| + \mu}$$

where μ is a parameter that is usually chosen to be the average length of documents in collection C.

Linear interpolation is used to combine multiple language models in order to estimate a new language model [45, 175]. In this case, $p(w|\theta_d)$ is given by:

$$p(w|\theta_d) = \sum_{i=1}^{L} \beta_i \; p(w|\theta_{d_i}) \tag{2.2}$$

where L is the number of language models, θ_{d_i} is the i^{th} representation of document D, and β_i is the weight associated with language model θ_{d_i}. The weights β_i are constrained to sum to 1, so that:

$$\sum_{i=1}^{L} \beta_i = 1, \text{ and } \beta_i \geq 0 \text{ for all } 1 \leq i \leq L$$

For every language model θ_{d_i} of the i^{th} document representation, the probability distribution $p(w|\theta_{d_i})$ is estimated using Equation (2.1). In this case, the smoothing is applied to the collection of i^{th} representation of documents, denoted by C_i. So, $p(w|\theta_{d_i})$ is given by:

$$p(w|\theta_{d_i}) = \alpha_{1i} \; p(w|D_i) + \alpha_{2i} \; p(w|C_i) \tag{2.3}$$

where D_i is the i^{th} representation of document D, α_{1i} and α_{2i} are Dirichlet prior smoothing parameters for the i^{th} representation of document D. In Chapter 5, language models are used for unsupervised baselines of dataset search.

2.2 Convolutional Neural Network (CNN)

A CNN [132] extracts features from data by defining a set of filters or kernels that spatially connect local regions. Compared to the dense networks, each neuron is connected to only a small number of neurons instead of being connected to all neurons from the previous layer. This design significantly reduces the number of parameters in the model. In addition, the weights in CNN filters are shared among multiple local regions for the input, and this further reduces the number of parameters. The outputs of the CNN are called feature maps. Pooling operations, such as average and max pooling, are usually applied to the feature map to keep only the significant signals, and to further reduce the dimensionality. A CNN kernel has a predefined size so that in order to handle the information in the border of the input, padding is introduced to enlarge the input. CNNs were first introduced to solve image-related tasks such as image classification [126, 216, 230, 103, 231], and were later adapted to solve text-related tasks such as NLP and information retrieval [46, 51, 99, 104, 130, 157, 178, 234, 111, 100]. In Chapter 7, CNN are used in our proposed neural architecture for learning to rank data tables.

2.3 Recurrent Neural Network (RNN)

An RNN [66] learns features and long-term dependencies from sequential and time-series data. RNN reads the input sequence sequentially to produce a hidden state in each timestamp. These hidden states can be seen as memory-cells that store the sequence information. A current hidden state is a function of the previous hidden state and the current input. Therefore, a hidden state is computed for each timestamp, and the hidden state that corresponds to the last timestamp in the sequence captures the context-aware representation of the entire sequence. Two major problems of the vanilla RNN are the vanishing and exploding gradients [183, 227] that can occur after back-propagation through time during the training phase. For example, for long sequences, when the gradient flows from later to earlier timestamps in the input sequence, the signal from the gradient can become very weak or vanish. Two variants of RNN which are LSTM and GRU have been proposed to capture long-term dependencies better than RNN, and therefore reduce the vanishing and exploding of the gradient. The new structures of LSTM and GRU allow the network to capture long-range dependencies. In Chapter 7, RNN are used in multiple supervised baselines for dataset search.

2.4 Long Short-Term Memory (LSTM)

Exploding and vanishing gradients during the training phase result in the failure of the network to learn long-term dependencies in the input data. LSTM [93] was introduced to mitigate the effects of exploding and vanishing gradients. LSTM differs from the vanilla RNN in the structure of the memory cell where three gates are introduced to control the information in the memory cell. First, the input gate controls the influence of the current input on the memory cell. Second, the forget gate controls the previous information that should be forgotten in the current timestamp. Third, the output gate controls the influence of the memory cell on the hidden state of the current timestamp. Compared to RNN, LSTM has led to significant improvements in multiple fields with sequential data such as text, video, and speech. LSTM has been applied to solve multiple tasks including language modeling [118], text classification [49], machine translation [228], video analysis [217], image captioning [113], and speech recognition [79]. In Chapter 7 and Chapter 10, LSTM is used in multiple baselines for dataset search and semantic labeling, respectively.

2.5 Gated Recurrent Units (GRU)

Similar to LSTM, GRU [39] is used for sequence-based tasks to capture long-term dependencies. However, unlike LSTM, GRU does not have separate memory cells. In LSTM, the output gate is used to control the memory content that is used by other units in the network. On the other hand, the GRU model does not contain an output gate, and therefore uses its content without any gating control. In addition, while LSTM computes the value of the new added memory independently of the forget gate, GRU does not independently control the new added activation but uses the reset gate to control the previous hidden state. More differences and similarities between LSTM and GRU are summarized by [43]. This model has been shown to achieve good performance in multiple tasks such as machine translation [39] and sentiment classification [233]. In Chapter 7 and Chapter 11, GRU are used in multiple baselines for dataset search and entity matching, respectively.

2.6 Word embedding

Words are embedded into low dimensional real-valued vectors based on the distributional hypothesis [87]. In many proposed models, the context is defined as the words that precede and follow a given target word in a fixed window [6, 162, 158, 186]. Mikolov et al. [159] proposed the Skip-gram model which scales to a corpora with billions of words. These pre-trained word embeddings are a key component in multiple models in neural language understanding and information retrieval tasks. However, there are multiple challenges for learning word embeddings. First, the embedding should capture the syntax and semantics of tokens. Second, the embedding should model the polysemy characteristic where a given word can have different meanings depending on the context. Multiple works have been proposed to produce context-sensitive embeddings for tokens that capture not only the meaning of a token, but also the contextual information of a token. Researchers have investigated the use of RNN to produce context-dependent representations for tokens [272, 152, 129]. The word embeddings are initialized using pre-trained embeddings, then the parameters of RNN are learned using labeled data from a given task. Peters et al. [187] proposed a semi-supervised approach to train a context-sensitive embedding using a neural language model pre-trained on large and unlabeled corpus. A forward and backward RNN are used to predict the next token so that the neural language model encodes both the semantic and syntactic features of tokens in a context. Adding pre-trained context-sensitive embeddings from both the forward and backward RNN improves the performance of the sequence tagging task. However, the embeddings are not deep, in the sense that they are not a function of all of the internal layers of both the forward and background language models. Recently, researchers have focused on creating deep context-sensitive embeddings such as ELMo [188] and BERT [58] which are trained on large amounts of unlabeled data and achieve high performance in multiple NLP tasks. In Chapter 5, the Skip-gram model is used to learn a new embedding for dataset search.

2.7 Attention mechanism

The attention mechanism was first proposed by Bahdanau et al. [3] for neural machine translation. The original Seq2Seq model [228] used an LSTM to encode a sentence from its source language and another LSTM to decode the sentence into a target language. However, this approach was unable to capture long-term dependencies. In order to solve this problem, Bahdanau et al. [3] proposed to simultaneously learn to align and translate the text. They learn attention weights which can produce context vectors that focus on a set of positions in a source sentence when predicting a target word. The attention vector is computed using a weighted sum of all the hidden states of an input sequence, where a given attention weight indicates the importance of a token from the source sequence in the attention vector of a token from the output sequence. Although introduced for machine translation, the attention mechanism has been a useful tool in many tasks such as document retrieval [157], document classification [273], sentiment classification [258], recommender systems [279], sentence semantic matching [289, 232, 257, 278], recognizing textual entailment [203], speech recognition [25], natural language understanding [27], and visual question-answering [150]. In Chapter 7, the attention mechanism is incorporated in our proposed method and multiple baselines, and in Chapter 11, the attention mechanism is used in entity matching.

2.8 Deep contextualized language models

Peters et al. [188] proposed ELMo which is a deep contextualized language model composed of forward and backward LSTMs. ELMo produces deep embeddings, where the representations from multiple layers of both the forward and backward LSTM are linearly combined using task-specific learnable weights to compute the embedding of a given token. Combining the internal states leads to richer embeddings. Although ELMo improves the results of multiple NLP tasks, ELMo decouples the left-to-right and right-to-left contexts by using a shallow concatenation of internal states from independently trained forward and backward LSTMs. Devlin et al. [58] proposed a language model, called Bidirectional Encoder Representations from Transformers (BERT), that fuses both left and right context.

BERT [58] is a deep contextualized language model that contains multiple layers of Transformer [246] blocks. Each Transformer block has a multi-head self-attention structure followed by a feed-forward network, and it outputs contextualized embeddings or hidden states for each token in the input. BERT is trained on unlabeled data over two pre-training tasks which are the masked language model, and next sentence prediction. Then, BERT can be used for downstream tasks on single text or text pairs using special tokens ([SEP] and [CLS]) that are added into the input. For single text classification, BERT special tokens, [CLS] and [SEP], are added to the beginning and the end of the input sequence, respectively. For applications that involve text pairs, BERT encodes the text pairs with bidirectional cross attention between the two sentences. In this case, the text pair is concatenated using [SEP], and then treated by BERT as a single text.

The sentence pair classification setting is used to solve multiple tasks in information retrieval including document retrieval [50, 173, 271], frequently asked question retrieval [205], passage re-ranking [172], and table retrieval [36]. The single sentence setting is used for text classification [224, 281]. BERT takes the final hidden state $\boldsymbol{h_\theta}$ of the first token [CLS] as the representation of the whole input sequence, where θ denotes the parameters of BERT. Then, a simple softmax layer, with parameters W, is added on top of BERT to predict the probability of a given label l:

$$p(l \mid \boldsymbol{h_\theta}) = \text{softmax}(W\boldsymbol{h_\theta}) \tag{2.4}$$

The parameters of BERT, denoted by θ, and the softmax layer parameters W are fine-tuned by maximizing the log-probability of the true label. BERT is used in Chapter 8 for dataset search, Chapter 10 for semantic labeling, and Chapter 11 for entity matching.

2.9 Neural ranking models for document retrieval

2.9.1 Overview

Ranking and retrieving documents that are relevant to a user's query is a classic information retrieval task. Given a query, the ranking model outputs a ranked list of documents so that the top ranked items should be more relevant to the user's query. Search engines are examples of systems that implement ad-hoc retrieval where the possible number of queries that are continually submitted to the system is huge. The general flowchart of document retrieval with neural ranking models is illustrated in Figure 2.1. A large collection of documents is indexed for a fast retrieval. A user enters a text-based query which goes through a query processing step consisting of a query reformulation and expansion [2]. Many neural ranking models have complex architectures, therefore computing the query-document relevance score using the neural ranking model for every document in the initial large collection of documents leads to a significant increase in the latency for obtaining a ranked list of documents from the user's side. So, the neural ranking component is usually used as a re-ranking step that takes two inputs which are the candidate documents and the processed query. The candidate documents are obtained from an unsupervised ranking stage, such as BM25 [202], which takes as inputs the initial set of indexed documents and the processed query. During the unsupervised ranking, the recall is more important than the precision to cover all possible relevant documents and forward a set of candidate documents, that has both relevant and irrelevant documents, to the neural based re-ranking stage. The output of the ranking model is a set of relevant documents to the user's query which are returned to the user in a particular order. In Chapter 9, we discuss a similar multi-stage model for dataset search to improve the efficiency of the ranking stage.

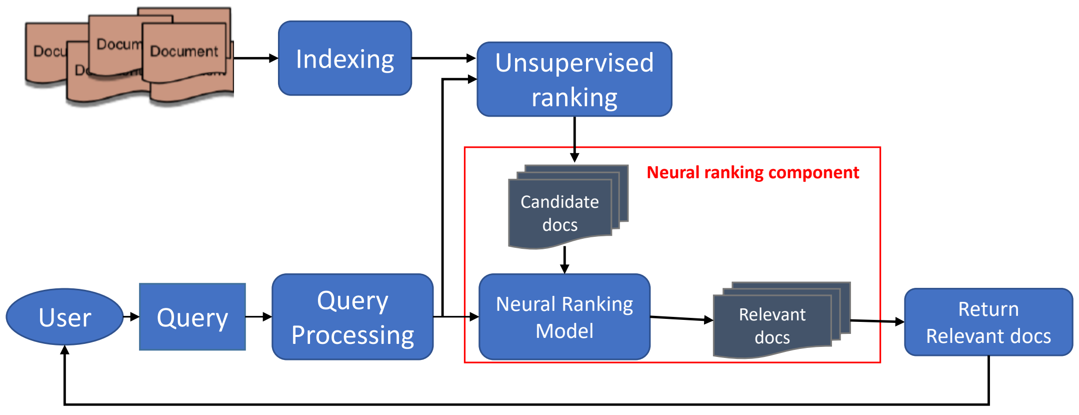


Figure 2.1: Overview of the flowchart of the neural ranking based document retrieval. The neural ranking component is highlighted within the red box. The inputs to the neural ranking model are the processed query and the candidate documents that are obtained from the traditional ranking phase. The final output of the neural ranking model is a ranking of relevant documents to the user's query.

The inputs to neural ranking models consist of queries and documents with variable lengths in which the ranking model usually faces a short query with keywords, and long documents from different authors with a large vocabulary. Although exact matching is an important signal in retrieval tasks, ranking models also need to semantically match queries and documents in order to accommodate vocabulary mismatch. In ad-hoc retrieval, features can be extracted from documents, queries, and document-query interactions. Some document features go beyond text content and can include number of incoming links, page rank, metadata, etc. A challenging scenario for a ranking model is to predict the relevance score by only using the document's textual content, because there is no guarantee to have additional features when ranking documents. Neural ranking models have been used to extract feature representations for the query and document using text data. For example, a deep neural network model can be used to map the query and documents to feature vectors independently, and then a relevance score is calculated using the extracted features. For query-document interaction, classic information retrieval models like BM25 can be considered as a query-document feature. For neural ranking models with a textual input for the query and document, features are extracted from the local interactions between the query and document. In chapter 7, we show the importance of extracting interaction-based features between the query and document in an early stage in the neural ranking model.

2.9.2 Task formulation

For ranking tasks, the objective is to output a ranked list of documents given a query representing an information need. Neural ranking models are trained using the LTR framework that starts with a phase to train a model to predict the relevance score from a given query-document pair. During the training phase, a set of queries $Q = \{q^1, q^2, \dots, q^{|Q|}\}$ and a large collection of documents $\mathcal{D}$ are provided. Without loss of generality, we suppose that the number of tokens in a given query is n, and the number of tokens in a given document is m. The groundtruth relevance scores for query-document pairs are needed to train the neural ranking model. In the general setting, for a given query, the groundtruth relevance scores are only known for a subset of documents from the large collection of documents $\mathcal{D}$. So, we formally define that each query q^i is associated with a subset of documents $d^i = (d^i_1, d^i_2, \dots, d^i_{l^i})$ from $\mathcal{D}$, where d^i_j denotes the j^{th} document for the i^{th} query, and l^i is the size of d^i. Each list of documents d^i is associated with a list of relevance scores $y^i = (y^i_1, y^i_2, \ldots, y^i_{l^i})$ where y^i_j denotes the relevance score of document d^i_j with respect to query q^i. The objective is to train a function f_w, with parameters w, that is used to predict the relevance score $z^i_j = f_w(q^i, d^i_j)$ of a given query-document pair (q^i, d^i_j). The function f_w is trained by minimizing a loss function $L(w)$. In LTR, the learning categories are grouped into three groups based on their training objectives: the pointwise, pairwise, and listwise approaches. In general, f_w is considered as the composition of two functions M and F, with F is a feature extractor function, and M is a ranking model. So, for a given query-document pair (q^i, d^i_j), z^i_j is given by:

$$z^i_j = f_w(q^i, d^i_j) = M \circ F(q^i, d^i_j) \tag{2.5}$$

In traditional ranking models, the function F represents a set of hand-crafted features. The set of hand-crafted features include query, document, and query-document features. A ranking model M is trained to map the feature vector $F(q^i, d^i_j)$ into a real-valued relevance score such that the most relevant documents to a given query are scored higher to maximize a rank-based metric. In recently proposed ranking models, deep learning architectures are leveraged to learn both feature vectors and models simultaneously. The features are extracted from query, document, and query-document interactions. The neural ranking models are trained using ground truth relevance of query-document pairs. In this dissertation, we use the LTR formulation to train our supervised models in dataset search.

2.9.3 Categories of strategies for learning for ad-hoc retrieval

Liu [144] divided LTR approaches into three categories based on their training objectives. In the pointwise category, each query-document pair is associated with a real-valued relevance score, and the objective of the training is to make a prediction of the exact relevance score using existing classification [75, 137] or regression models [47, 71]. However, predicting the exact relevance score may not be necessary because the final objective is to produce a ranked list of documents. In Chapter 8, we use the pointwise loss to train our proposed model.

In the pairwise category, the ranking model does not try to accurately predict the exact real-valued relevance score of a query-document pair; instead, the objective of the training is to focus on the relative order between two documents for a given query. So, by training using the pairwise category, the ranking model tries to produce a ranked list of documents. In the pairwise approach, ranking is reduced to a binary classification to predict which of two documents is more relevant to a given query. Many pairwise approaches are proposed in the literature including methods that are based on support vector machines [90, 109], neural networks [16], Boosting [69], and other machine learning algorithms [299, 298]. For a given query, the number of pairs is quadratic, which means that if there is an imbalance in the relevance judgments where more groundtruth relevance scores are available for a particular query, this imbalance will be magnified by the pairwise approach. In addition, the pairwise method is more sensitive to noise than the pointwise method because a noisy relevance score for a single query-document pair leads to multiple mislabeled document pairs.

The third learning category for ad-hoc retrieval is known as the listwise category, proposed by Cao et al. [21]. In the listwise category, the input to the ranking model is the entire set of documents that are associated with a given query in the training data. Listwise approaches can be divided into two types. In the first type, the loss function is directly related to evaluation measures [24, 26, 193]. So, they directly optimize for a ranking metric such as NDCG, which is more challenging because these metrics are often not differentiable with respect to the model parameters. Therefore, these metrics are relaxed by approximation to make computation efficient. For the second type, the loss function is differentiable, but it is not directly related to the evaluation measures [21, 96, 249]. For example, in ListNet [21], the probability distribution of permutations is used to define the loss function. Since a ranking list can be seen as a permutation of documents associated with a given query, a model representing the probability distribution of permutations, like the Plackett-Luce [191] model, can be applied for ranking in ListNet. In Chapter 6 and Chapter 7, we use the listwise loss function to train our proposed supervised models for dataset search.

2.9.4 Deep learning-based models

When extracting features from a query-document pair, the feature extractor F can be applied separately to the query and document [214, 97, 194, 213, 167, 176, 251], or it can be applied to the interaction between the query and document [252, 89, 270, 67, 95]. The first type is known as the representation-focused model, and it tries to extract a good feature representation for a single text using a deep neural network. For example, Shen et al. [214] proposed a Convolutional Deep Structured Semantic Models (C-DSSM) in which a convolutional neural network (CNN) is used instead of feed-forward-networks in Siamese [15] architecture. So, the feature extractor F is a CNN, while M is the cosine similarity function. In ARC-I [95], F is a CNN, and M is a multi-layer perceptron (MLP). The models that belong to the first category of deep matching architectures differ the interaction between two sentences until learning individual representations, so that there is a risk of losing important details for the matching task. The second type, the interaction-based models, starts by building local interactions between two texts based on basic representations, then trains a deep model to capture the important interaction patterns for matching. For example, in ARC-II [95], F maps each text to a sequence of word embeddings, while M is a CNN over the interaction matrix between the two texts. In Chapter 7 and Chapter 8, we show the importance of interaction-based architectures in dataset search.

The neural ranking models in document retrieval present two important matching techniques: semantic matching and relevance matching [80]. Semantic matching is introduced in multiple text matching tasks, such as natural language inference, and paraphrase identification. Semantic matching, which aims to model the semantic similarity between the query and the document, assumes that the input texts are homogeneous. Semantic matching captures composition and grammar information to match two input texts which are compared in their entirety. In information retrieval, the Question-Answering (QA) task is a good scenario for semantic matching, where semantic and syntactic features are important to compute the relevance score. On the other hand, semantic matching is not enough for document retrieval, because a typical scenario is to have a query that contains keywords. In such cases, the relevance matching is needed to achieve better retrieval results. Relevance matching is introduced by Guo et al. [80] to solve the case of heterogeneous query and document in ad hoc document retrieval. The query can be expressed by keywords, so a semantic signal is less informative in this case because the composition and grammar of a keyword-based query are not well defined. In addition, the position of a given token in a query has less importance than the strength of the similarity signal, so some neural ranking models, like DRMM [80], do not preserve the position information when computing the query-document feature vector. An important signal in the relevance matching is the exact matching of query and document tokens. In traditional retrieval models, like BM25, exact matching is primarily used to rank a set of documents, and the model works reasonably well as an initial ranker. Incorporating exact matching into neural ranking models can improve the retrieval performance mainly in terms of recall for keyword-based queries because as in traditional ad hoc document retrieval, the document has more content than the query and the presence of query keywords in a document is an initial indicator of relevance. In Chapter 7, we incorporate semantic and relevance matching signals in our proposed neural ranking model to improve the evaluation metrics of dataset search.

2.9.5 Deep contextualized language model-based document retrieval

The sentence pair classification setting is used to solve the document retrieval task. The overview of BERT for the document retrieval is shown in Figure 2.2. In general, the input sequence to BERT is composed of the query q and selected tokens s_d from the document d: [[CLS], q, [SEP], s_d, [SEP]]. The selected tokens can be the whole document, sentences, passages, or individual tokens. The hidden state of the [CLS] token is used for the final retrieval score prediction. In Chapter 10 and Chapter 11, we use the sentence pair classification setting in semantic labeling and entity matching, respectively.

While BERT has been successfully applied to QA, applying BERT to ad-hoc retrieval of documents comes with the challenge of having significantly longer documents than BERT allows (BERT cannot take input sequences longer than 512 tokens). Yang et al. [271] proposed to address the length limit challenge by dividing documents into sentences and applying BERT to each of these sentences. The sentence-level representation of a document is motivated by recent work [288] which shows that a single excerpt of a document is better than a full document for high recall in retrieval. In addition, using sentence-level representation is related to research in passage-level document ranking [146]. For each document, its relevance to the query can be predicted using the maximum relevance of its component sentences, which is denoted as the best sentence. Yang et al. [271] generalize the best sentence concept by choosing the top-k sentences from each document based on the retrieval score calculated by BERT for sentence pair classification setting. A weighted sum of the top-k sentence-level scores,

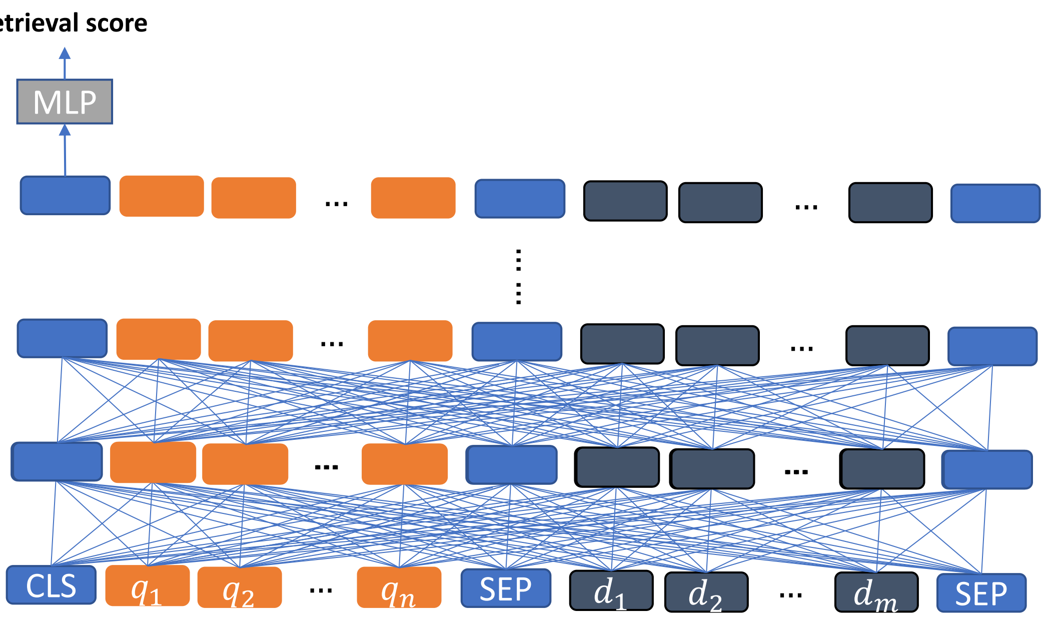


Figure 2.2: Overview of BERT model for document retrieval. The input sequence to BERT is composed of the query $q = q_1 q_2 \dots q_n$ (shown with orange color in the first layer) and selected tokens $s_d = d_1 d_2 \dots d_m$ (shown with dark blue color in the first layer) from the document d. The BERT-based model is formed of multiple layers of Transformer blocks where each token attends to all tokens in the input sequence for all layers. From the second to the last layer, each cell represents the hidden state of the corresponding token which is obtained from the Transformer. The query and document tokens are concatenated using the [SEP] token, and [CLS] is added to the beginning of the concatenated sequence. The hidden state of the [CLS] token is used as input to MLP in order to predict the relevance score.

which are computed by BERT, is then applied to predict the retrieval score of the query-document pair. In Chapter 8, we show how to incorporate the sentence-level representation into our proposed BERT-based model for dataset search by dividing a given dataset into rows and columns.

In order to capture both relevance and semantic matching, MacAvaney et al. [153] propose a joint model that incorporates the representation of [CLS] from the query-document pair into existing neural ranking models (DRMM [80], PACCR [99], and K-NRM [262]). The representation of the [CLS] token provides a strong semantic matching signal given that BERT is pretrained on the next-sentence prediction. As we explained previously, some of the neural ranking models, like DRMM, capture relevance matching for each query term based on the similarities with the document tokens. For ranking models, MacAvaney et al. [153] use pretrained contextual language representations as input, instead of the conventional pretrained word vectors to produce a context-aware representation for each token from the query and document. In Chapter 8, we show how to incorporate the context-aware embeddings from BERT into our proposed model to encode the structure- and context-aware sequences, that are obtained from fusing the structural and textual information of datasets.

Nogueira et al. [173] propose a multi-stage ranking architecture. The first stage consists of extracting the candidate documents using BM25. In this stage, the recall is more important than the precision to cover all possible relevant documents. The irrelevant documents can be discarded in the next stages. The second stage, called monoBERT, uses a pointwise ranking strategy to filter the candidate documents from the first stage. The classification setting of BERT with sentence pairs is used to compute the relevance scores. The third stage, called duoBERT, is a pairwise learning strategy that computes the probability of a given document being more relevant than another candidate document. Documents from the second stage are ranked using duoBERT relevance scores in order to obtain the final ranked list of documents. The input to duoBERT is the concatenation of query, first document, and second document, where [SEP] is added between the sentences, and [CLS] is added to the beginning of the concatenated sentence. In Chapter 9, we show how to incorporate our proposed BERT-based model for dataset search into a multi-stage ranking architecture to reduce the memory and time complexity.

2.10 Knowledge graph embeddings

Various methods have been proposed for representation learning of knowledge graphs, which aims to project entities and relations into a continuous space. TransE [14], inspired by Word2Vec [158], is the most representative translation-based model, which considers the translation operation between head and tail entities for relations. The variants of TransE, such as TransH [259] and TransR [142], follow a similar principle but use different scoring functions to learn the embeddings. Socher et al. [219] apply neural tensor networks to learn knowledge graph embeddings. Dettmers et al. [57] propose a convolutional neural network approach to learn knowledge graph embeddings and use them to perform link prediction. RDF2Vec [201] adapts the Word2Vec [158] approach to RDF (Resource Description Framework) graphs in order to learn embeddings for entities in RDF graphs.

The recent success of graph neural networks has boosted research on various tasks. R-GCN [208] pioneered the use of graph convolutional networks to model relations in knowledge graphs. The embeddings learned by R-GCN have shown to be effective for downstream tasks such as entity classification and link prediction. R-GCN [208] can be seen as an extension of GCN [60, 121] for relational data that operates on a local graph neighborhood using a message-passing framework [77]. The R-GCN model updates the hidden representation of node v_i in the relational graph as given by:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \tag{2.6}$$

where $h_i^{(l)} \in R^{d(l)}$ is the l-th layer hidden state of node v_i in the neural network, $d(l)$ is the dimension of the embedding in the l-th layer, and $\sigma(.)$ is a nonlinear activation function. $\mathcal{N}_i^r$ denotes the set of r-neighbors, where $r \in \mathcal{R}$. $c_{i,r}$ is a normalization constant that is equal to $|N_i^r|$. Unlike the linear transformation in GCN that can be applied to any node in a given layer, R-GCN has a relation-specific linear transformation, denoted by $W_r^{(l)}$, that depends both on the type and direction of the edge in a directed and labeled graph. $W_0^{(l)}$ is a trainable matrix that incorporates the l-th layer representation into the $l+1$-th layer of the neural network. R-GCN is formed of multiple stacked layers with non-linear activation functions to capture complex patterns in the graph that are not only related to direct neighbors. Updating the nodes in R-GCN for a given layer is done in parallel to reduce computation time.

More recently, Xu et al. [265] first construct a product knowledge graph and then propose a self-attention-enhanced distributed representation learning method with an efficient multi-task training schema to learn the graph embeddings, which can improve the performance of downstream tasks such as search ranking and recommendation. In chapter 6, the knowledge graph is used to represent datasets and learn new embeddings for tokens of datasets.

2.11 Structured prediction

Structured prediction [196, 55, 128, 174] is the task of learning a function that maps the input to a structured output that is not a simple discrete or Boolean variable, but can be more complicated with rich substructure. Therefore, the mapping function predicts a structured label such as a label for each token in a sequence for natural language parsing, or a label for each pixel or region in an image for the image segmentation task. The exponential size of the possible structured labels leads to computational challenges in both training and inference phases that can be NP-Hard as stated by Deshwal et al. [55].

Multiple methods have been proposed to learn the mapping function in the structured prediction setting such as Conditional Random Fields (CRF) [127], and structured Support Vector Machines [128]. Deep learning models have been leveraged in structured prediction to capture the structural dependencies between the inputs and outputs. For example, in part-of-speech (POS) tagging which consists of labeling each token in a text with a POS tag (noun, adjective, verb, etc), a deep neural network based model, called BI-LSTM-CRF [98], has been proposed to solve the POS tagging task. The structural dependency between the inputs and outputs is captured using both the Bi-LSTM features and the CRF that finds the best tagging sequence. Deep learning is also integrated in structured prediction for computer vision applications [174, 196]. For example, Quattoni et al. [196] formulated annotating images with semantic tuples as a structured prediction task where CRF models are used to map the feature vectors that are extracted using convolutional neural networks to semantic tuples. In Chapter 10, we formulate the semantic labeling task using the structured prediction setting.

2.12 Domain adaptation

Domain adaptation (DA) studies the transfer of task knowledge from a single or multiple labeled source domains to an unlabeled target domain. In this dissertation, we are interested in the case of multiple source domains known as Multi-Source DA (MSDA). Using only unlabeled data from the target domain is known as Unsupervised DA (UDA).

Existing approaches in UDA focus on reducing the domain shift between the source and target domains by aligning feature vectors [5, 177]. Representation learning methods have been proposed for UDA such as domain adversarial networks [296, 211]. Other representation learning methods include comparing the marginal distribution between the source and target domains in an adversarial way [81] and minimizing the covariance between the source and target representations [223]. An effective strategy in the case of MSDA is known as a mixture of experts [81, 119, 261]. Kim et al. [119] proposed to incorporate an attention mechanism to combine the predictions from multiple models trained on the source domains. Guo et al. [81] proposed a method that is based on a mixture of experts where the posteriors of the models are combined using a point-to-set Mahalanobis distance metric between an input sample and source domains. Wright and Augenstein [261] improved the performance of a mixture of expert-based models using deep contextualized language models (DCLM) as experts in source domains. This work follows a line of research that investigates the use of Transformer-based models in DA [83, 86, 151, 200]. Ma et al. [151] improved the performance of BERT in the target domain for natural language inference by incorporating a similarity of target domain to source domains with curriculum learning [7]. AdaptaBERT [86] is a BERT-based model that is proposed in the case of UDA for the sequence labeling by adding a masked language modeling in the target domain. Fine-tuning of BERT on the target domain was also shown to be effective in the sentiment analysis task [200]. Gururangan et al. [83] combines both domain and task adaptive pretraining to improve the performance of RoBERTa on multiple NLP tasks. The task-adaptive pretraining represents pretraining on unlabeled datasets that are relevant to the task by continuing pretraining RoBERTa on these datasets. In Chapter 11, domain adaptation is used in our proposed method for entity matching.

Chapter 3

Tasks Definitions and Related Work

3.1 Table Search and Similarity

In table retrieval, a table can be considered as a document, and traditional document retrieval methods can be applied to table ranking. Cafarella et al. [17, 18] retrieve relevant documents using web search engines, and then tables are extracted from the highest-ranked documents. The simplest approach is to represent a table by a single field containing all the text associated with the table. The retrieval score is then calculated using existing retrieval methods, such as language models or BM25 [202]. However, a table has multiple components of varying importance, which means that retrieval models for multi-field documents are often more appropriate [190]. Pimplikar and Sarawagi [190] proposed a late fusion [292] method for multi-field ranking. In this case, for a given query, a score is calculated independently for every field, and then a linear combination of the scores is taken. The final score is given by:

$$score(Q, T) = \sum_{i} w_i \times score(Q, f_i) \tag{3.1}$$

where Q is a given query, T is a table, f_i is the i^{th} field of T, and w_i is the weight associated with f_i. For supervised ranking of tables, multiple query, table, and query-table features are proposed in the literature [18, 10]. Zhang and Balog [293] proposed extending these features with semantic matching between queries and tables using semantic spaces including: Word embeddings, Graph embeddings, Bag-of-entities and Bag-of-categories. DBpedia [134] is used to construct a vector of zeros and ones for both bag-of-entities and bag-of-categories. The dimension of bag-of-entities is equal to the total number of entities in the knowledge base, where a value of 1 indicates that the entity is mentioned in the table. The same applies to bag-of-categories with a dimension that is equal to the total number of Wikipedia categories.A supervised model is then trained using the semantic and traditional features.

Recent works have used embedding techniques to learn a low dimensional representation for table tokens. Deng et al. [290] proposed a natural language modeling-based approach to create embeddings for table tokens. The trained embedding is then used with entity similarity from a knowledge base to rank tables. Using matrix factorization, Chen et al. [34] generated additional headers that are used in ranking table-query pairs. The authors showed that the generated headers improve the performance of unsupervised table retrieval.

Deep contextualized language models, like BERT [58] and RoBERTa [148], have been recently proposed to solve natural language understanding [253, 145] and information retrieval [276, 50, 173, 271, 205, 172] tasks. Different from traditional word embeddings, the pre-trained neural language models are contextual with the representation of a token is a function of the entire sentence. This is mainly achieved using a self-attention structure called Transformer [246]. Building on BERT, Chen et al. [36] proposed a BERT-based ranking model to capture the matching signals between the query and the table fields using the sentence pair setting. They first select the most salient items of a table to construct the BERT representation, where different types of table items and salient signals are tested.

Shraga et al. [215] use neural networks to learn unimodal features of a table which are combined into a multimodal representation. The final table-query relevance is estimated based on the query representation and multimodal representation. Tables can also be represented as graphs to solve table retrieval [256, 37].

Table similarity consists of predicting the semantic similarity between tables and then classifying a table pair as similar or dissimilar. Das Sarma et al. [53] proposed a table similarity method that is based on entity consistency and expansion, and schema similarity, and is used to find related tables in a large corpus of heterogeneous data. In entity consistency, similar tables should have similar entities, and in entity expansion, the queried table should add new entities to the query table. In schema similarity, similar tables should have similar schemas that represent similar entities. Relevance scores are computed for entity consistency and expansion, and schema similarity, and then combined to predict the entity complement score of a table pair.

Deep learning models have been leveraged to predict the similarity score between tables. TabSim [85] treats the data table fields independently in which one bi-LSTM model is used to map the caption of a data table to an embedding vector, and a second attention-based model is used to compute the embeddings of the columns of a data table. The caption and column embeddings are concatenated to form the representation of a data table. After extracting the feature vectors of two data tables, Euclidean distance is used to predict the semantic similarity between tables.

3.2 Semantic Labeling

Semantic labeling [244, 204, 101] consists of classifying sets of data values into a predefined set or categories known as semantic labels. These approaches rely on a multiclass classification setup where the labels are manually defined and curated. Hulsebos et al. [101] extend the set of semantic types by considering 275 DBpedia [1] properties. These manually defined concepts, like *Birth place*, *Continent*, and *Product*, represent the semantic types that are frequently found in datasets. In order to infer the semantic type of a column using data values, the authors defined multiple categories of hand-crafted features. Each feature category has a different performance and noise level, so that the authors propose a multi-input neural networks model, instead of simply concatenating all features, and feeding the resulting feature vector to a single-input neural network. The multi-input neural networks model is composed of multiple identical subnetworks without weights sharing. Each subnetwork consists of two fully connected hidden layers with batch normalization, rectified linear unit (ReLU) activation functions, and dropout.

Knowledge base-based methods [28, 29] integrate DBpedia [1] to predict semantic labels, where entities from DBpedia that match all the column cells are used as additional information for a given column values. Semantic types use a limited set of vocabulary, and can restrict the number of categories that can be considered when inferring the label of a given column. In practice, the predefined set of semantic types may not apply for new datasets. Chen et al. [33] proposed a schema label generation task, in which the objective is to infer the *schema label*, and not only the semantic type. This setting can be seen as a multiclass classification task, where each column's label in the training set represents a possible semantic label. Generating schema labels is more challenging as the number of possible labels is large compared to the predefined set of semantic types. The authors extract hand-crafted features from data values to predict schema labels. The set of features include content and unique content ratio [61], and the content histogram which is a 20-dimensional vector extracted using fast Fourier transform (FFT). Random forest classifier is used to predict schema labels from the curated features.

Schema matching is related to semantic type detection where the objective is to find correspondence between attributes in different schemas. Existing data on the Web, such as WebTables [19], and knowledge bases, such as DBPedia [1] and Freebase [13], are used in schema matching. Syed et al. [229] use headers and data values to predict the class of a column in the target ontology or knowledge base. The data values provide additional information that can disambiguate between the possible candidates. Limaye et al. [141] associate one or more types from YAGO [222] with each attribute or column in the table using a probabilistic graphical model. Another probabilistic approach, that is based on the maximum likelihood hypothesis, is introduced by Venetis et al. [247]. The best label is chosen to maximize the probability of the values given the class label for a given column. The authors showed that class labels that are automatically extracted from the web provide more coverage for column's labeling than using manually created knowledge bases like YAGO [222] and Freebase [13].

Matching functions are used to infer the correct semantic labels for data values. Pham et al. [189] treat semantic labeling as a combination of many binary classification problems. After extracting similarity metrics features from a pair of attributes, each feature vector is given a True/False label, where True means that the attributes have the same semantic type, and False indicates that the attributes are not sharing the same semantic type. Logistic Regression and Random Forests are used to predict the matching score. For the similarity metrics features, the authors investigated multiple metrics, such as Jaccard similarity [155], cosine similarity of the product of term frequency (TF) and inverse document frequency (IDF), known as TF-IDF [155], Kolmogorov-Smirnov test (KS test) [133], and Mann-Whitney test (MW test) [133].

Mueller and Smola [166] proposed a neural network embedding for data values to predict the matching score of two sets of data values. The matching score is estimated using the distance between the embeddings of two sets of data values. The score is adjusted using the output of another neural network to distinguish two columns that are different but their data values are identically distributed.

Semantic type prediction is formalized as a ranking problem in the approach proposed by Ramnandan et al. [199]. Training data values are considered as documents, and the previously-unseen data values are considered as queries. So, in the prediction phase, the objective is to extract the top k candidate semantic labels for the new data values by ranking semantic labels in decreasing order using cosine similarity between a query feature and every document feature in the training data. The authors distinguished between textual and numeric data. For textual data, the feature vector is a weighted bag of words with TF-IDF. For numerical data, the authors used a statistical hypothesis testing to analyze the distribution of numerical data values that correspond to a given semantic label. The statistical hypothesis test is performed between each sample in the training data and the testing sample. The returned p-values are then ranked in descending order to predict the top k candidate semantic labels for the testing data values.

3.3 Entity matching

Entity matching (EM) [165, 139, 114, 62, 4] is the field of research that solves the problem of finding records that refer to the same real-world entity. This task, also known as data matching, record linkage, entity resolution, etc, has been intensively studied in recent years because of the importance of EM in data cleaning and integration. Given two collections of records D_1 and D_2, EM classifies a pair of entities $(e_1, e_2), \forall e_1 \in D_1, e_2 \in D_2$ into match or non-match. The records from D_1 and D_2 can have the same or different set of attributes. The value of each attribute is composed of a sequence of tokens. In figure 3.1, we show examples of pairs of records for EM from the Amazon-Google dataset where in both subfigures the above record is from Amazon and the below record is from Google. In Figure 3.1(a), both records refer to the same real-world entity *adobe photoshop 4.0* although in one record the manufacturer value is missing, and the prices are different. In figure 3.1(b), the difference in the value of *title* attribute in both records clearly indicates that records refer to different entities.

Comparing all record pairs from D_1 and D_2 grows quadratically, and it becomes

title	manufacturer	price
adobe photoshop elements 4.0 (mac)	adobe	89.99

title	manufacturer	price
adobe photoshop elements 4.0 photo-editing software for mac	NULL	85.95

(a) Records refer to the same entity: adobe photoshop 4.0

title	manufacturer	price
microsoft publisher 2007 version upgrade	microsoft	99.95

title	manufacturer	price
microsoft excel 2007 (pc)	microsoft	229.95

(b) Records refer to different entities

Figure 3.1: Examples of pair of records for EM from the Amazon-Google dataset. The above record is from Amazon and the below is from Google.

very time-consuming to predict the matching records for the input datasets. Therefore, a set of candidate pairs $C \subset D_1 \times D_2$, where $|C| \ll |D_1 \times D_2|$ is selected in a separate step, called blocking, before running a computationally expensive algorithm for EM. Multiple blocking methods have been proposed in the literature [42, 68, 181]. After the blocking step, each record pair $(e_1, e_2) \in C$ is compared to predict a binary label indicating a match or non-match. Prior works have proposed string similarity-based methods to compare records [41, 65, 149]. Traditional supervised classifiers, such as decision trees, support vector machines, and naive Bayes have been proposed to map the string similarities-based feature vector to a binary class label [40, 11]. In addition, rule-based methods have been proposed to solve EM [52, 64, 218]. Recently, deep learning (DL)-based methods have been proposed to solve EM [62, 70, 114, 165, 297, 139]. The DL methods of EM can be categorized as attribute- and record-level comparison methods. Attribute comparators predict the label of a pair of records based on the signals collected from matching values of the same attribute. DeepMatcher[165], which is the SOTA attribute-level comparator, explores multiple techniques to compute the attribute representation from word embedding, where combining both bidirectional GRU and decomposable attention [182] leads to the best results. FastText [12] is used for word embedding in DeepMatcher.

The SOTA method in EM is a record-based comparator known as Ditto [139] which is based on DCLM. Ditto models each record by alternating between attributes and data values with two additional special tokens [COL] and [VAL]. Incorporating attribute names in the record representation provides the Transformer [246] layers with more information to match attributes of two records. Then, Ditto adapts the sentence pair classification setting to EM in order to compare record pairs using the special tokens [SEP] and [CLS] that are added into the input. In addition, Ditto explores domain-specific optimizations by injecting domain knowledge into the input in the form of span typing and normalization. Ditto uses data augmentation techniques during the training phase with span-, attribute-, and record-level operators consisting of deletion, shuffling, and swapping.

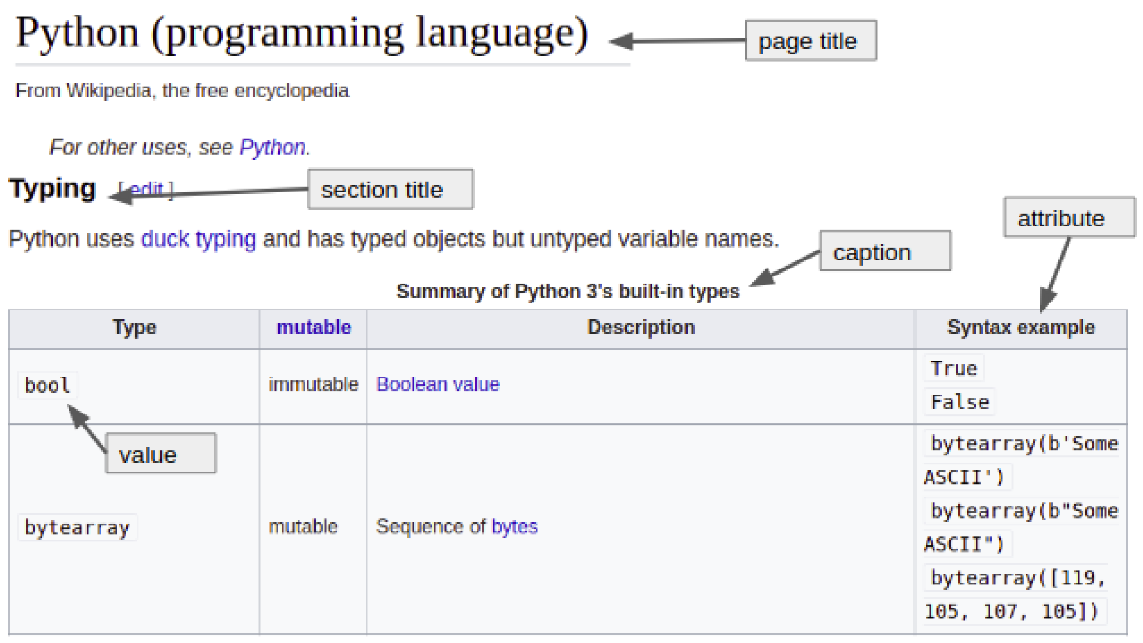
Chapter 4

Data Collections

4.1 Table Retrieval

4.1.1 WikiTables

This dataset is composed of the WikiTables corpus [9] containing over 1.6M tables. Each table has five indexable fields: table caption, attributes (column headings), data rows, page title, and section title. An example of a table from Wikipedia is shown in Figure 4.1. In addition, each table contains statistics which are: number of columns,



Python (programming language)

From Wikipedia, the free encyclopedia

For other uses, see Python.

Typing [edit]

Python uses duck typing and has typed objects but untyped variable names.

Summary of Python 3's built-in types

Type	mutable	Description	Syntax example
`bool`	immutable	Boolean value	`True` `False`
`bytearray`	mutable	Sequence of bytes	`bytearray(b'Some ASCII')` `bytearray(b"Some ASCII")` `bytearray([119, 105, 107, 105])`

Figure 4.1: Example of table from a Wikipedia page.

number of rows, and set of numerical columns of the table. Additional LTR features [293] are provided for each table. The set of LTR features include number of in-links to the page embedding the table, number of out-links from the page embedding the table, number of page views, etc. We use the same test queries that were used by Zhang and Balog [293]. The queries are divided into two subsets: the first subset contains queries collected by Cafarella et al. [17] using Amazon's Mechanical Turk platform, and a second subset contains queries collected by Venetis et al. [248] using Google Squared. The second subset consists of queries created by combining the name of an 'instance class' and a property (i.e., attribute).

We present three statistics about the query collection: the minimum length of a query is 1 term, the maximum length of a query is 7 terms, and the average length of the queries in the collection is 2.8 terms. Examples of queries are: 'world interest rates Table', 'fuel consumption', 'state capitals and largest cities in us', 'baseball teams captain', etc. We use Zhang and Balog's [293] ground truth of query-table relevance, where every query-table pair is evaluated using three numbers: 0 means irrelevant, 1 means partially relevant and 2 means relevant. The objective of the annotators was to use the retrieved tables to create a new table that fulfills the query. So, for a given query, they needed to find tables that are useful in forming a single table that matches the query. By using this task to evaluate a given table's relevance, if a table could not be used to create the final table, it is given a relevance 0. If only some values are used from the table, it is partially relevant. Finally, if blocks of a table are used, it is considered relevant. There are 60 queries in the WikiTables collection, and the number of query-table pairs is equal to 3117.

4.1.2 WebQueryTable

WebQueryTable[1] collection is introduced by Yan et al. [268]. Unlike the WikiTables collection that contains tables only from Wikipedia, WebQueryTable is composed of more various tables collected from web pages. The total number of tables in WebQueryTable is $297,884$. Each table has four indexable fields: table caption, table subcaption, attributes (column headings), and data rows. In addition, WebQueryTable [268] contains $21,142$ queries. Each query-table has a relevance value that is

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

equal to 0 or 1, and only one table is relevant to a given query. The tables were selected from the top ranked web page, and the annotators were asked to label whether the extracted table is relevant to the query or not. The minimum length of a query is 1 term, the maximum length of a query is 22 terms, and the average length of the queries in the collection is 4.6 terms. Examples of queries are: ‘large big mac meal price australia’,‘largest cities in USA area wise’, and ‘largest financial institutions’.

4.1.3 Query by example data

Zhang and Balog [291] proposed a query by table dataset that is composed of 50 Wikipedia tables used as input queries. The query tables are related to multiple topics, and each table has at least five rows and three columns. For the ground truth relevance scores of table pairs, each pair is evaluated using three numbers: 2 means highly relevant and it indicates that the queried table is about the same topic of the query table with additional content, 1 means relevant and it indicates that the queried table contains a content that largely overlaps with the query table, and 0 means irrelevant. The total number of table pairs is 2850.

4.2 Table Similarity

4.2.1 WikiTables for table similarity

In addition to keyword-based table retrieval, we adapt WikiTables for table similarity. As in TabSim [85], we iterate over all the queries of WikiTables, and if two tables are relevant to a query, the table pair is given a label 1. On the other hand, an irrelevant table to a query is considered not similar to all tables that are relevant to the same query, and therefore the table pair is given label 0.

4.2.2 PMC

Habibi et al. [85] proposed a table corpus that is formed from the PubMed Central (PMC) Open Access subset, and used for evaluation on the table similarity task. This collection is related to biomedicine and life sciences. Each table contains a caption and data values. The table pairs are annotated for binary classification by comparing the caption and data values of each table. A given table pair is given a label dissimilar if both data values and captions are labeled dissimilar, otherwise the table pair is given the label similar. In the PMC table corpus, there are 1391 table pairs, where 542 pairs are similar and 849 pairs are dissimilar.

4.3 Semantic Labeling

4.3.1 WikiTables for semantic labeling

This dataset is composed of the WikiTables corpus [9] which contains over $1.6M$ tables that are extracted from Wikipedia. Since a lot of tables have unexpected formats, we preprocess tables so that we only keep tables that have enough content with at least 3 columns and 50 rows. We further filter the columns whose schema labels appear less than 10 times in the table corpus, as there are not enough data tables that can be used to train the model to recognize these labels. We experiment on $15,252$ data tables, with a total number of $82,981$ columns. The total number of schema labels is $1,088$.

4.3.2 Log Tables from Network Equipment

Our work is partially motivated by the business need to automatically generate schema labels for the data tables extracted from log files of network equipment. Network log files contain computer-generated event records, such as authentication attempts, process assessment calls and information output of network equipment, and are instrumental for network performance monitoring and fault diagnosis.

For the purpose of schema label auto-generation, we shall utilize the existing data tables that have already been collected in an internal platform used by network care engineers from parsing log files. In the current pipeline, engineers design a parser for each type of log file, and these parsers generate the tables. We have collected 329 tables from this platform with log files coming from products on wireless equipment such as base stations, Radio Access Network and Radio Network Controllers. To evaluate our methods on header prediction, we removed tables that have less than 10 rows and cleaned up columns that have mostly invalid values (such as NULL, empty string, or NA). The remaining set contains 248 tables. The number of rows of these tables have a very skewed distribution with quantiles being 138 (25%), 551 (50%)

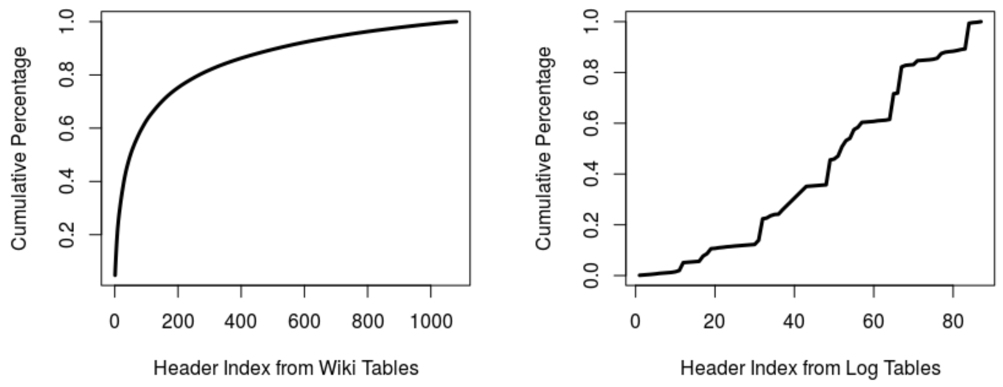


Figure 4.2: Cumulative Frequency Distribution of Headers

and 1954 (75%), while the number of columns ranges from 3 to 48 with many tables having in the neighborhood of 10 columns. For our purpose, we focus on 87 headers from these tables that have more than 3 instances.

Figure 4.2 shows the cumulative frequency distribution for the headers from the WikiTables and Log Tables datasets, from the most to the least popular. There is a small set of labels that are much more frequently occurring in WikiTables. One reason that the labels in log tables are more scattered is because the tables are manually collected from diverse products as we would like to understand the performance of our algorithm in various situations.

4.3.3 Combined data

We also evaluate our method using two web table collections which are: T2Dv2[2] and Efthymiou [63]. We combine these two datasets into a single dataset that contains 395 data tables. The data set has $1,739$ total columns and 166 distinct schema labels.

[2]http://webdatacommons.org/webtables/goldstandardV2.html

Table 4.1: Datasets for entity matching experiments.

Dataset	Domain	Size	% matches	nb attributes
Shoes	clothing	5,805	21.95	1
Cameras	electronics	5,255	22.03	1
Computers	electronics	8,094	22.42	1
Watches	electronics	6,413	22.85	1
DBLP-GoogleScholar	citation	28,707	18.62	4
DBLP-ACM	citation	12,363	17.95	4
Fodors-Zagats	restaurant	946	11.62	6
Beer	product	450	15.11	4
iTunes-Amazon	music	539	24.48	8
Abt-Buy	product	9,575	10.73	3
Amazon-Google	software	11,460	10.18	3
Walmart-Amazon	electronics	10,242	9.39	5

4.4 Entity Matching

Table 4.1 represents all the datasets that we use in our experiments. The 12 datasets are from the entity resolution Benchmark datasets [124] and the Magellan data repository [123]. These datasets cover multiple domains including clothing, electronics, citation, restaurant, products, music, and software. Each dataset is composed of candidate pairs of records from two structured tables that have the same set of attributes. The datasets vary in the size and this simulates real-world scenarios where there are some domains that are more frequent than others. The total number of attributes in all datasets ranges from 1 to 8. The rate of matches in all datasets ranges from 9.39% to 24.48%. Clearly, there is a class imbalance in all datasets where the non-matching class is significantly larger than the matching class. Each dataset is split into training, validation, and testing, and we use the same pre-splited datasets in Ditto [139].

Chapter 5

Multiple Context Embedding for Attributes of Data Tables

5.1 Introduction

Words are embedded into low dimensional real-valued vectors based on the distributional hypothesis [87]. In many models, the context is defined as the words that precede and follow a given target word in a fixed window [158, 185]. Mikolov et al. [159] proposed the Skip-gram model which scales to a corpora with billions of words. In order to solve the entity linking problem, Gentile et al. [74] proposed an embedding model, instead of using the classic bag-of-word representation. They create sentences from the attributes and/or values of the table, then use these sentences to train new Skip-gram embeddings for the tokens. The similarity between tables is calculated using cosine similarity between the learned embeddings for table tokens. In their approach to table classification, Ghasemi-Gol and Szekely [76] proposed a new unsupervised embedding for tables. They define four different contexts for each cell value: text within each cell, text in the corresponding attribute or header, text in adjacent cells, and text surrounding the table in the web page.

In a table retrieval task, a table can be considered as a document, and traditional document retrieval methods can be applied to table ranking. Cafarella et al. [17, 18] retrieve relevant documents using web search engines, and then tables are extracted from the highest-ranking retrieved documents. The simplest approach is to represent a table by a single field containing all the text associated with the table. The retrieval score is then calculated using existing retrieval methods, such as language models or BM25 [202]. However, a table has multiple components of varying importance, which means that retrieval models for multi-field documents are often more appropriate.

A major drawback of prior approaches is that they are based on pretrained embeddings, and they ignore contextual information within tables in the ranking step. In order to overcome the limitations of using pre-trained embeddings for table search, we propose learning a new model for word embeddings of attribute tokens that is used to predict the contextual information of tables in the ranking phase.

Our proposed approach has three stages: in the first step, we build word embeddings for attribute tokens using contextual information from each table. In the second step, for a given attribute token of a table, we predict its context using the trained contextual model and augment the table with this additional, implicit, and descriptive information. In the third step, we calculate a score for a given query-table pair in order to rank and retrieve tables that are related to the query. We use a mixed ranking model that incorporates the metadata of a table and the additional predicted context in order to calculate the retrieval score.

Our work differs from several table embedding methods in the literature. Unlike Gentile et al. [74], our contexts do not depend on the arbitrary ordering of rows or columns in the dataset, and we learn a word embedding for attribute tokens by enlarging the context to cover metadata of tables. The additional contexts are useful in table retrieval as more predicted contexts are available when scoring a table against a query using multi-field ranking approaches. In Nishida et al. [171], a word embedding of every cell token is obtained after supervised training of a hybrid model. In our case, we use a different architecture with unsupervised training when learning embeddings. Similar to Ghasemi-Gol and Szekely [76], we identify multiple contexts from the table. However, a key difference is that our model distinguishes between the different contexts, rather than treating them uniformly. Also, we have different notions of context: we do not simply treat the four cells adjacent to a cell as context, and the only locality information that is used is that all cells in the header row are considered context for each other.

5.2 Word embeddings for attribute tokens

The objective is to learn an embedding for every attribute token that captures its contextual information inside a table. We use an adapted Skip-gram model [158] for tables. The training objective is to learn a real-valued representation for attribute tokens using contextual information from each table. Then, the trained model is used to predict additional tokens that are relevant to the table.

Our model differs from the original Skip-gram model in two major aspects: training context and vocabulary. In the original Skip-gram model, the context is based on the surrounding words of a given word. However, in a table, how to define context is not so obvious. Clearly, other tokens in the name of an attribute count as context, but are there other meaningful contexts? The metadata of a table clearly provides a larger context for the attributes, and an attribute is also contextualized by the other attributes that appear in the same table. We also argue that the cell values provide meaningful contextual information. For example, an attribute that consisted of 50 two-character values including 'AZ', 'KY', 'MS', 'WA' , etc. should be assigned an embedding that is similar to another attribute with the same set of values, even if the attributes shared no name tokens. Thus, the context for an attribute token is rich, but we argue that these different types of contexts should not all be treated uniformly. Levy and Goldberg [136] have shown that, for word embeddings, differentiating contexts can lead to embeddings that better express similarity as opposed to relatedness. Inspired by this work, we use a multi-context model, so that a token that appears in the metadata has a different impact than the same token when it appears in a cell value. Thus, for a given attribute's token, we have four different types of contexts: description, values, other tokens from the same attribute, and other tokens from attributes in the same table. For simplicity, we append a distinct suffix to every context token in order to distinguish between the different contexts in the training data.

Our model uses different input and output dictionaries: the input dictionary contains tokens of attributes extracted from the collection of tables, and the output dictionary is composed of tokens from all four types of contexts. More formally, given a sequence of T training attribute tokens $a_1, a_2, a_3, \ldots, a_T$, the objective of our model is to maximize the sum of log probabilities using our proposed contextual information:

$$\sum_{t=1}^{T}\Big(\sum_{w_d \in D_t} log\ p(w_d|a_t) + \sum_{w_v \in V_t} log\ p(w_v|a_t) \\ + \sum_{w_{oa} \in OA_t} log\ p(w_{oa}|a_t) + \sum_{w_{sa} \in SA_t} log\ p(w_{sa}|a_t)\Big)$$

where a_t is a given attribute's token, D_t is the set of description context tokens of a_t, V_t is the set of values context of a_t, OA_t is the set of other tokens from attributes in the same table containing a_t, and SA_t is the set of other tokens from same attribute containing a_t.

Given an output context w_c and input token a_t, the Skip-gram model defines $p(w_c|a_t)$ using the softmax function. Computing the softmax probability is expensive because it requires summing over all the words in the output dictionary. To address this problem, we estimate the softmax probability using Noise Contrastive Estimation (NCE) [84, 163]. NCE reduces the language model estimation problem to a binary logistic regression classifier that distinguishes between data and noise.

5.2.1 Managing Numerical Cell Values

Numerical values play a more prominent role in tables than in text documents. For example, 26.9% of cells in the WikiTables dataset are numeric values, while in a random sample of 372 tables from data.gov [33], 58.2% of the cells are numeric. An attribute's numeric values can contribute to its interpretation and thus provide useful context information: four digit numbers beginning with 19 or 20 are likely to be years, while in the United States five-digit numbers are often zip codes and sequences of the form *ddd-ddd-dddd* are often phone numbers. However, the number of unique numeric values in a dataset can be far larger than the number of unique non-numeric tokens, and adding them to the vocabulary will lead to an increase in the size of the output dictionary. For the WikiTables dataset, we obtain 2,401,425 unique tokens (including numbers) in the output vocabulary. Furthermore, context should ideally recognize that small numeric distances reflect similar contexts, and context should not be adversely impacted by noise or rounding errors. That is, the numbers 998 and 1002 are quite close, as are 3.14 and 3.14159. Likewise, two attributes with a range of values from 1960 to 2020 should be considered to have very similar values context even if less than half of the values appear in both attributes.

A typical approach to handling numbers in word embeddings is to replace each digit with a special character, such as the hash symbol (‘#’). For simplicity, we tokenize using the same punctuation that we use to tokenize the rest of the text. Thus, 999-99-9999 becomes the tokens ‘999’, ‘99’ and ‘9999’, and then ‘###’,‘##’ and ‘####’. This means that regardless of the number of distinct numerical values, we only add a number of tokens to the output dictionary that is less or equal to the length of the longest successive sequence of digits. As a partial solution to the problem of recognizing numbers that are close, while keeping the cardinality of numeric tokens low, we propose a new approach: keep the leading digit of a number, and only replace other digits by ‘#’ in order to refine numerical values context. With this approach, the size of output vocabulary decreases to 2,032,424, a reduction of about 15%. However, in other datasets, such as those in data.gov, numbers are more prevalent, and the savings will be more significant.

5.2.2 Table features

We describe a table using three sets of features:

- original description (D_o): Set of tokens extracted from the table metadata such as its title and/or caption,
- original attributes (A_o): Set of tokens extracted from the header row of the table, and
- original values (V_o): Set of tokens extracted from the data rows of the table (i.e., all rows other than the header).

We augment these features with additional features produced by our trained model. First, for an attribute token a in the input dictionary D_I, we predict the different contexts by extracting the top k words from the output dictionary that have the highest probabilities. We denote our set of top k contexts by C_k. We divide the predicted contexts into three categories: description context D_c, values context V_c, and attributes context A_c. More formally,

$$C_k = D_c \cup V_c \cup A_c$$

Second, the hidden layer $H \in \mathbb{R}^{W_i \times h}$ of our model, where W_i is the number of attributes tokens, presents a new word embedding of dimension h for each attribute token. We use the new word embedding to extract the set of top m closest tokens, denoted by I_m, to a given attribute token a, using cosine similarity.

The final predicted context P_c is given by:

$$P_c = C_k \cup I_m = D_c \cup V_c \cup A_c \cup I_m$$

5.2.3 Unsupervised Ranking methods for table retrieval

The predicted context P_c enables us to augment every table with additional fields that capture the contextual information obtained from the original attribute tokens. The additional contexts can be combined with original fields, such as title, data, original attributes, etc., in order to improve multi-field ranking, and thus table retrieval. We propose using the multiple ranking mechanisms based on Equation (3.1). We use traditional ranking methods such as BM25 and TF-IDF. The third ranking method, LM-Ranking, is based on combining language models [175]. In other words, we index tables using the contents of the original fields and additional predicted contexts, then we estimate a language model for every field, and we combine the estimated language models using Equation (2.2).

The fourth ranking method, Late-avg, is a late fusion similarity model [293] based on Equation (3.1) for multi-field ranking, but with scores calculated by averaging the cosine similarity between the embeddings of all pairs of query terms and terms of field f_i. Given an embedding $E(\cdot)$ for a term t:

$$score_{la}(Q, f_i) = \frac{1}{|Q| \times m_i} \sum_{k=1}^{|Q|} \sum_{j=1}^{m_i} cosine(E(q_k), E(t_{ji})) \tag{5.1}$$

where m_i is the length of field f_i, q_k is the k^{th} query term of Q, and t_{ji} is the j^{th} token in f_i.

Kenter and de Rijke [115] proposed a Semantic Text Similarity (STS)-based ranking method specifically intended for short texts. This combines a traditional BM25 formula with semantic similarity computed from word embeddings. Because the semantic similarity is computed on query token/field token pairs, this has aspects of a late-fusion approach. In STS, the query is assumed to be the short text.

As a simplification of the above approach, we consider a pure late-fusion approach that only considers the semantic similarity, and ignores the BM25 aspects. In this approach, we select the best matching query term for each table field term and refer to it as MaxQuery:

$$score_{mq}(Q, f_i) = \sum_{j=1}^{m_i} \max_{k \in [1,\ldots,|Q|]} cosine(E(q_k), E(t_{ji}))$$

Finally, we argue that Kenter and de Rijke's assumptions about short text similarity do not apply to the table retrieval problem. In particular, rather than choosing the best query token for each field token, we choose the best field token for each query token. Typically, the set of table tokens will be much larger than the set of query tokens. A table could be a good match for a query even if only a portion of the table is relevant. Our final model, MaxTable is a late fusion similarity model, but we find the closest table term to each query term using cosine similarity, and then sum over these similarities:

$$score_{mt}(Q, f_i) = \sum_{k=1}^{|Q|} \max_{j \in [1,\ldots,m_i]} cosine(E(q_k), E(t_{ji})) \tag{5.2}$$

In the above ranking methods, the choice of embedding $E(\cdot)$ depends on which field is being compared to the query. Since we only apply our embedding approach to attribute tokens, other fields will contain tokens for which we do not produce embeddings. For these fields, we use pre-trained fastText [12] embeddings, which are built from character-level n-grams, allowing embeddings to be created even for terms that have not been seen before. Specifically, we use our embeddings on the original attribute field A_o, the predicted context attribute field A_c, and the closest tokens I_m. All other fields (D_o, D_c, V_o, and V_c) use fastText embeddings for computing similarity. Recall that the predicted description contexts and value contexts were produced by our embedding model, so our approach still contributes to the scores even when fastText embeddings are used for determining cosine similarity.

5.3 Evaluation

5.3.1 Baselines

We compare the performance of our proposed model with two baselines:

Single-field document ranking: A table is considered as a single field document by concatenating a subset of WikiTables fields: table caption, attributes, data rows, page title and section title (as in [17, 18]). In the result tables, we identify which fields were used. Then a language model is estimated for the formed document in order to rank a given table against the query. We can also calculate the score of a query-table pair using word embedding-based ranking methods from Section 5.2.3.

Multi-field document ranking: In a multi-field ranking scenario, a table is defined using five fields: page title, section title, table caption, attributes and table body or values. We compare our method against two baselines in the category of multi-field document ranking using word embedding. The first multi-field ranking method, MultiField-P, is based on the pretrained fastText word embedding as in Zhang and Balog [293] when calculating cosine similarity. In the second multi-field ranking method, MultiField-G, we use a word embedding trained by the adapted Skip-gram model to the context introduced by Ghasemi-Gol and Szekely [76]. Field weights are optimized using grid search.

We conduct an ablation study that evaluates five variations of our model. The first variation is called SCON (single context), in which we treat all contexts as one context. In other words, we do not append a distinct suffix to every context token to distinguish between different contexts. The other four variations are based on different formulations of the values context V_t for a given attribute a_t. In the NOVAL variation, we ignore the values context by setting $V_t{=}\emptyset$. In the NONUM variation, V_t includes only string contexts. In the HASHNUM variation, we replace each digit with the hash symbol ('#'), but otherwise use all value tokens and multiple contexts. Finally, the MCON variation is our full model, where we keep the leading digit for numbers and replace other digits by '#'. In HASHNUM and MCON, V_t includes both numerical and string value contexts. In the experimental results section, we report results from all five variations.

Table 5.1: Example of contexts for token 'syntax' of web table shown in Figure 4.1

D_{syntax}	OA_{syntax}	SA_{syntax}	V_{syntax}
['syntax','python_d'], ['syntax','programming_d'], ['syntax','language_d'], ['syntax','typing_d'], ['syntax','summary_d'], ['syntax','python_d'] ['syntax','built_d'], ['syntax','types_d']	['syntax','mutable_o'], ['syntax','type_o'], ['syntax','description_o']	['syntax','example_s']	['syntax','true_v'], ['syntax','false_v'], ['syntax','bytearray_v'], ['syntax','some_v'], ['syntax','ascii_v'], ['syntax','bytearray_v'], ['syntax','some_v'], ['syntax','ascii_v'], ['syntax','bytearray_v'], ['syntax','1##_v'], ['syntax','1##_v'], ['syntax','1##_v'], ['syntax','1##_v']

5.3.2 Experimental Setup

We use the full set of $1,652,771$ tables to train our embedding model. For the description context D_t of a given attribute token a_t, we concatenate three fields from WikiTables: page title, section title and caption.

We set the dimension of word embeddings h to 100, and the number of labels used in NCE estimation to $10,000$. We train our model for 3 epochs with a batch size of 100. We use SGD to minimize the loss function, and update the weights of our model. We set the learning rate to 0.01. The model is implemented using TensorFlow, with Tesla T4 GPU (memory Clock Rate: 1.59 GHz). For context prediction, we set the size of C_k, k, and the size of I_m, m, to 20. In order to calculate the retrieval score for query-table pairs by combining language models, we use the implementation in Hasibi et al. [88] which is based on Elasticsearch.

5.3.3 Example of MCON context

We give an example of the extracted contexts from the Wikipedia table shown in Figure 4.1. We refer to the union of the page title, caption and section title as the description. Given the token 'syntax' of attribute 'Syntax example', we construct D_{syntax}, OA_{syntax}, SA_{syntax}, and V_{syntax}. We show the different target-context pairs in Table 5.1.

5.3.4 Model statistics

We start by comparing our models in terms of sizes of input dictionary, output dictionary, and training data (number of target-context pairs). The statistics are shown in Table 5.2. For all five variations, we have the same size of input vocabulary. For output vocabulary, MCON has the largest vocabulary, as it includes multiple contexts and the numerical context. The output dictionary size does not influence the training time because we use NCE to estimate the softmax probabilities. The number of parameters for MCON is significantly larger than NOVAL because of the difference in the size of output vocabulary. So, MCON allocates more memory than NOVAL to update model parameters. The training time is directly related to the number of target-context pairs. In our experiments using a single Tesla T4 GPU, the average training time for 4000 steps each with a batch size of 100 is 34.67 seconds. Our models are trained for three epochs. So, for MCON, we need 14.22 hours for one epoch of training and the total training time is 42.66 hours. Because training depends directly on the number of target-context pairs, NOVAL is significantly less, at 13.39 hours.

Table 5.2: Statistics of our models

Model	Input dict	Output dict	Target-context pairs
SCON	118,421	1,608,455	590,661,355
NOVAL	118,421	415,272	185,390,306
NONUM	118,421	1,997,633	432,705,346
HASHNUM	118,421	2,025,698	590,661,355
MCON	118,421	2,032,424	590,661,355

5.3.5 Semantic similarity for MCON word embedding

In Table 5.3, we show examples of using cosine similarity to extract tokens from the input vocabulary that are close to a given attribute token. In WikiTables data, the attribute token ‘pos’ is frequently present in tables that are related to sport. In general, ‘pos’ means the ranking position of a team or player. As shown in Table 5.3, the closest tokens to ‘pos’ are also related to sport. For example, we have ‘w’ which is an abbreviation of ‘win’, ‘l’ which is an abbreviation of ‘loss’, and also we can find a synonym of ‘pos’ which is the token ‘position’.

Table 5.3: Examples of cosine similarity between embeddings of tokens from our input vocabulary

Original attribute token	Closest attributes tokens from input vocabulary
'pos'	'w', 'l', 'position', 'rank', 'points', 'game', 'de', 'final', 'positions', 'total', 'results', 'team', 'round', 'place', 'pts'
'actor'	'director', 'role', 'writer', 'character', 'cast', 'artist', 'directed', 'written', 'title', 'name', 'production', 'winner', 'description', 'film', 'episode'
'height'	'weight', 'hometown', 'women', 'length', 'overall', 'college', 'us', 'remarks', 'singles', 'men', 'american', 'current', 'end', 'average', 'model'
'vote'	'seats', 'percentage', 'parties', 'democratic', 'people', 'change', 'non', 'canada', 'australia', 'previous', 'point', 'regional', 'seat', 'post', 'body'

5.3.6 Ranking results

We evaluate the performance of our proposed method and baselines on the table retrieval task using Normalized Discounted Cumulative Gain (NDCG) [106] at cut-off thresholds 5, 10, 15, and 20. All evaluation metrics results are reported using the TREC evaluation software, trec_eval[1].

Table 5.4: Table retrieval evaluation results using MaxTable

Method	**Fields**	**NDCG@5**	**NDCG@10**	**NDCG@15**	**NDCG@20**
Single-field document ranking	all	0.4715	0.4832	0.5155	0.5404
Single-field document ranking	cell values	0.3292	0.3775	0.4245	0.4657
Single-field document ranking	description	0.4632	0.4912	0.5330	0.5462
Single-field document ranking	attributes	0.3204	0.3545	0.4137	0.4584
MultiField-P	all	0.4794	0.4930	0.5298	0.5473
MultiField-G	all	0.4610	0.4818	0.5051	0.5386
SCON	all	0.4824	0.5022	0.5343	0.5494
NOVAL	all	0.4813	0.5021	0.5323	0.5491
NONUM	all	0.4862	0.5037	0.5368	0.5505
HASHNUM	all	0.4902	0.5043	0.5367	0.5505
MCON	all	**0.5088**	**0.5117**	**0.5460**	**0.5587**

In Table 5.4, we show the NDCG results of table retrieval using the MaxTable ranking method (results for other ranking methods showed similar trends). We show that MultiField-P leads to a better performance than the single-field document ranking. From the results of the single-field document ranking using only cell values, we observe that the cell value-based single-field document ranking is not effective in

[1]https://trec.nist.gov/trec_eval/trec_eval.8.1.tar.gz

ranking query-table pairs. The additional predicted contexts using our MCON model have the best performance for all NDCG metrics.

In Table 5.5, we summarize the results of all ranking approaches from Section 5.2.3 using our best model, MCON. Here, the MaxTable ranking method is shown to be the most effective ranking for our embedding and context prediction approach. Aggregating scores using the sum of maximum similarities is more effective than using average similarity. Late-avg is stricter as it requires a query term to have large similarity with multiple tokens of a given field in order to obtain a high score. On the other hand, for MaxTable, a high similarity between a query term and one token from a given field is enough to obtain a high similarity score for a given field in a table. Among the ranking methods that are not based on word embedding (LM-Ranking, BM25, and TF-IDF), LM-Ranking achieves higher performance at all NDCG cut-off thresholds. Note, that for our table features, STS actually performs worse than plain BM25, while simply using its approach to semantic similarity gets closer to BM25.

Table 5.5: MCON table retrieval results

Method	NDCG@5	NDCG@10	NDCG@15	NDCG@20
BM25	0.4545	0.4854	0.5186	0.5449
TF-IDF	0.4316	0.4746	0.5073	0.5344
LM-Ranking	0.4755	0.4976	0.5316	0.5548
Late-avg	0.4740	0.5025	0.5241	0.5464
STS	0.4323	0.4502	0.4863	0.5158
MaxQuery	0.4642	0.4726	0.5087	0.5310
MaxTable	**0.5088**	**0.5117**	**0.5460**	**0.5587**

5.3.7 Analysis of query subsets

In Table 5.6, we show NDCG@5 for every subset of queries using MCON and MultiField-P with MaxTable ranking method. Both methods have better performance on queries subset 1 than subset 2. Overall, we achieve better results than MultiField-P in both subsets.

We show a more fine-grained analysis by plotting query-level performance in both subsets. We plot the query-level difference in NDCG@5 between MCON and MultiField-P using the MaxTable ranking method. In Figure 5.1(a), the rightmost bar corresponds to the query 'composition of the sun'. For this query, there is one

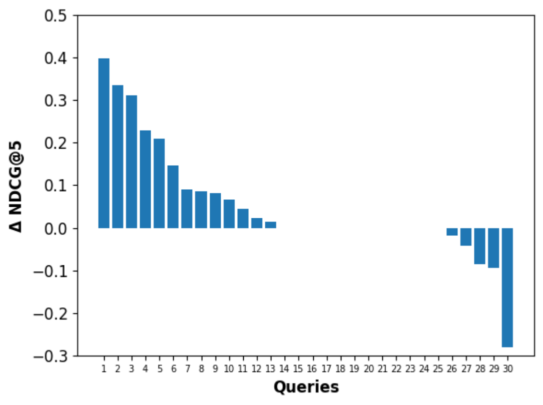
Table 5.6: Table retrieval results, MCON vs MultiField-P, on the two query subsets in term of NDCG@5 using MaxTable

Method	**Subset 1**	**Subset 2**
MCON	**0.5939**	**0.4908**
MultiField-P	0.5603	0.4570

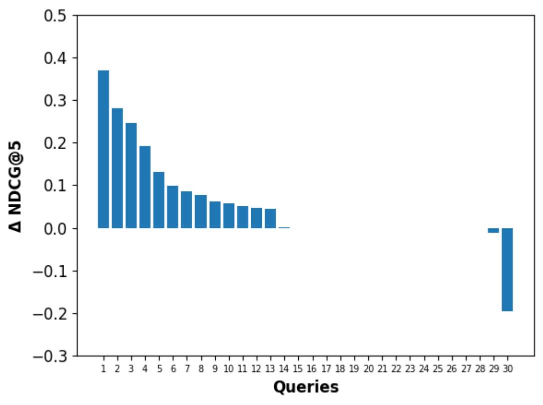
query-table pair with rank 2, one query-table pair with rank 1, and the other pairs are irrelevant (rank 0). MCON returns many irrelevant tables so that we obtain a low NDCG@5 score for this specific query. The table that is relevant to the query belongs to a Wikipedia page that has the title 'Atmosphere of Jupiter', the section title is 'Chemical composition', and the caption is 'Elemental abundances relative to hydrogen in Jupiter and Sun'. This table is ranked second using our approach. The top ranked table using our approach is from the same Wikipedia page and section, and has the caption 'Isotopic ratios in Jupiter and Sun'. Although this table has zero relevance in the ground truth, we believe that this table is indeed relevant to the query. The table that is ranked third by our algorithm belongs to 'Political composition'. The "somehow relevant" table is ranked 9^{th} by our method. For these two cases, the irrelevant predicted contexts affect the ranking negatively, and this can be explained by the lack of tables in training data that are related to the query's topic. In Figure 5.1(b), the rightmost bar corresponds to the query 'broadway musicals director'. In the testing collection, there is only one query-table pair that is "somehow relevant", and all other pairs are irrelevant. In the test data, the "somehow relevant" table has inaccurate attributes (as verified by examining the original Wikipedia page), leading our model to predict contexts that are not relevant to the actual table, and lowering the query-table similarity score.

5.4 Summary

We have shown that using multiple, differentiated contexts can result in more useful attribute embeddings. When the MaxTable ranking method is used for the table retrieval task, our MCON system has up to 5.47% improvement in NDCG@5 over a method that uses the same context fields but treats them as the same context. Likewise, we have shown that the data values of an attribute provide useful context information: our full system always performs better than the version that does not



(a) Queries subset 1



(b) Queries subset 2

Figure 5.1: Query-level NDCG@5 difference between MCON and MultiField-P

include values as context by as much as 5.71% in NDCG@5. Finally, we have shown that our simple treatment of numeric values also leads to better embeddings: when numeric values are dropped, NDCG@5 scores drop by up to 4.44%.

MCON embeddings can be used as an initial unsupervised ranker to extract an initial set of relevant tables to the user's query, where the recall is more important than the precision. This is similar to the document retrieval where BM25 is used in many cases as an initial ranker. Our results show that MCON has better performance than BM25 and language models, therefore MCON can be used to obtain a more relevant initial set of tables.

MCON embeddings capture mainly the co-occurrence information obtained from the adapted Skip-gram model. In addition, MCON provides embeddings only for the dataset attributes. To capture richer contextual information of a table corpus, in the next chapter we represent the table corpus as a knowledge graph. With the graph representation, multiple signals can be incorporated when learning the embedding of tokens in datasets. In document retrieval, researchers have focused on using learning-to-rank methods for document retrieval [293, 254, 263, 161] to refine the initial set of extracted documents. Similar to document retrieval, we incorporate the graph embedding into a new learning-to-rank architecture for dataset search to refine the initial set of relevant tables.

Chapter 6

Graph Embedding for Data Tables

6.1 Introduction

We address multiple research questions in this chapter. The first research question (**RQ1**) is how to capture multiple signals, such as the semantic and lexical information, when learning the embedding of tokens in datasets. Inspired by recent progress of transfer learning on graph neural networks, we propose an heterogeneous graph that incorporates multiple signals. The second research question (**RQ2**) consists of how to learn an embedding for each node in the heterogeneous graph. After learning the graph-based embedding, the third research question (**RQ3**) consists of how to incorporate the graph-based embeddings into an LTR model that improves table retrieval results. So, our proposed approach is a two-phased table retrieval method which uses graph embeddings pretrained on a large table corpus, denoted as Multiple Embeddings R-GCN (MultiEm-RGCN).

6.2 Knowledge Graph Construction and Embedding Learning

RQ1 is answered by phase I, where we construct a knowledge graph containing two types of knowledge: dataset-dependent knowledge and dataset-agnostic knowledge. The graph contains words and tables from a collection of tables as nodes. To incorporate dataset-dependent knowledge, point-wise mutual information (PMI) between word nodes, and term frequency-inverse document frequency (TF-IDF) between table and word nodes are computed based on the collection of tables. To avoid overfitting, we incorporate dataset-agnostic knowledge via external resources containing semantic knowledge from pretrained word embeddings, and lexical knowledge from WordNet. This incorporates a subgraph from WordNet into our knowledge graph. We model the graph with R-GCN which is used to learn multiple types of embeddings simultaneously. In phase II, we solve the table retrieval task by incorporating the R-GCN heterogeneous embeddings from phase I into a new learning-to-rank (LTR) architecture that combines multiple embedding spaces into one joint model.

We denote a knowledge graph by $G = (V, T, R)$, with a set of nodes V, a set of relation types R, and a set of directed edges $(v_i, r, v_j) \in T$, where v_i,$v_j \in V$ and $r \in R$. T can be seen as an RDF collection that contains (s, p, o) triples representing the knowledge graph. In this section, we first give a brief overview of R-GCN. Then we introduce how to construct a knowledge graph (KG) based on the table corpus given a set of predefined relations, which incorporate dataset-dependent knowledge and dataset-agnostic knowledge. After that, we describe how to learn high-quality node embeddings based on the constructed KG with link prediction as the pretraining task under the R-GCN framework. Ultimately, we present a Multiple Embeddings R-GCN (MultiEm-RGCN) model with two phases: phase I consists of training unsupervised embedding using R-GCN, and phase II consists of incorporating the multiple embeddings into a new LTR model.

6.2.1 Heterogeneous Knowledge Graph Construction

We describe how to build a meaningful knowledge graph for a large collection of tables that captures both general knowledge and dataset specific knowledge. Our graph contains word nodes and table nodes. The word nodes are constructed from the table collection and external resources. For a given word in the table collection, we also use its synonyms defined in WordNet and the hypernyms of these synonyms, which also have corresponding nodes in our constructed knowledge graph. In other words, our final graph includes a subset of WordNet relevant to the table collection.

We construct edges that encode two types of knowledge: ***dataset-dependent knowledge*** and ***dataset-agnostic knowledge***. For dataset-dependent knowledge, we build table-word edges and word-word edges from the table collection. However, only pretraining node embeddings from such a graph could overfit the dataset collection and harm the generalization ability of learned node embeddings, especially given a small training set. Therefore, we also propose to encode dataset-agnostic knowledge from external resources such as other pretrained word embeddings and WordNet. By constructing edges that encode both dataset-dependent knowledge and dataset-agnostic knowledge, we assume the learned node embeddings can capture both dataset specific information and open world knowledge. The overview of our proposed knowledge graph is shown in Figure 6.1. We build our graph G using RDF triples T. Initially, T is empty, and in this section we show how to build T.

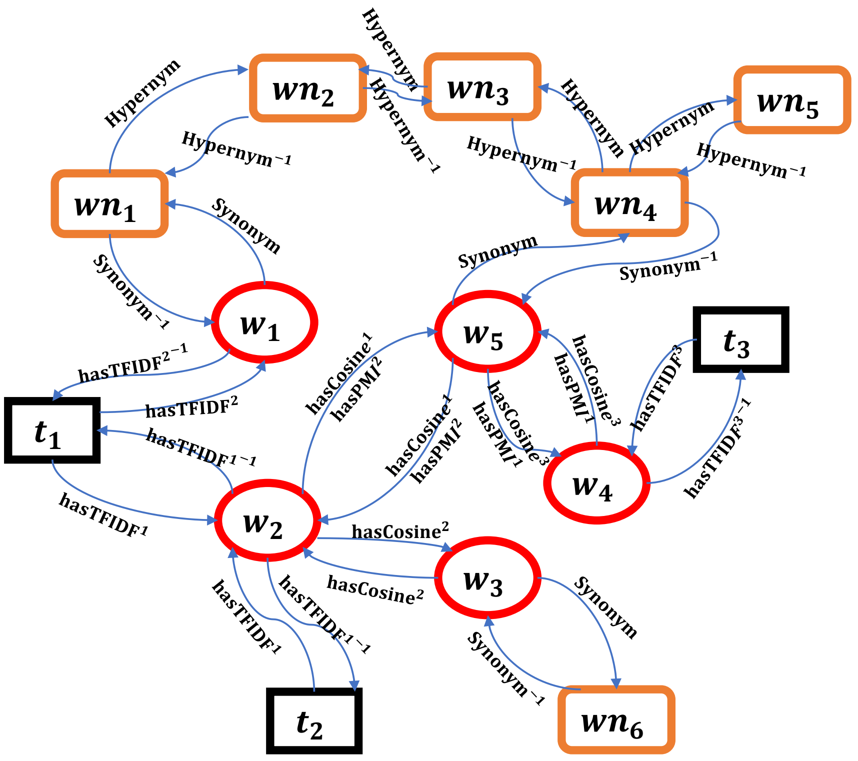


Figure 6.1: Overview of phase I of the proposed method MultiEm-RGCN. We use the same edge for $hasCosine$ and $hasPMI$ to avoid clutter in the graph. We can notice that the words w_2 and w_3 have only the $hasCosine^2$ relation because w_2 and w_3 do not co-occur in the tables collection.

Dataset-agnostic knowledge: We consider two types of dataset-agnostic knowledge. The first is ***semantic knowledge*** from word embeddings, such as Glove [186], pretrained on a large corpus. The cosine similarity of a word pair can be treated as prior information for two word nodes in our graph.

Let $d_{semantic}(w_i, w_j)$ denote the cosine similarity between two words w_i and w_j, which is a real value in the interval $[-1, 1]$. When building the knowledge graph, we do not keep the exact value of $d_{semantic}(w_i, w_j)$. Instead, we define several relations that represent different levels of similarities. This idea is inspired by the histogram matching method by Guo et al. [80] used for query document matching. We discretize the interval into a set of ordered bins and each bin has a corresponding edge type. Suppose that W is the set of word tokens in the training collection. We build the triples with p_{cosine} as the predicate only for those word pairs whose cosine similarity is larger than a threshold M_{cos}, since we care more about the semantic similarities rather than the dissimilarities, and the inferred dissimilarities from word embeddings could be inaccurate and trivial (two words randomly selected are likely to be dissimilar). In order to reduce the effect of extreme values which may result in bins with few data points, we calculate the mean m_{cos} and standard deviation std_{cos} of the set of all valid cosine similarities. Then we set the interval as $[min_{cos}, max_{cos}]$ where min_{cos} is the smallest cosine similarity that is larger than $m_{cos} - 2 \times std_{cos}$, and max_{cos} is the largest cosine similarity that is smaller than $m_{cos} + 2 \times std_{cos}$. We linearly divide $[min_{cos}, max_{cos}]$ into n_{cos} intervals and the k-th interval has a corresponding predicate $hasCosine^k$. For example, if $d_{semantic}(w_i, w_j)$ belongs to the k-th interval, then we add $(w_i, hasCosine^k, w_j)$ to T. Note that there are word pairs that have cosine similarity smaller than min_{cos}: we assign them to the 1st bin, and those have cosine similarity larger than max_{cos} are assigned to the last bin. Here, we define the set of semantic relation triples as

$$SemT = \{(w_i, hasCosine^k, w_j) | w_i, w_j \in W \text{ and } d_{semantic}(w_i, w_j) > M_{cos}\} \tag{6.1}$$

where $d_{semantic}(w_i, w_j)$ belongs to the k-th interval. We add $SemT$ to T.

The second type of dataset-agnostic knowledge incorporated into our graph is ***lexical knowledge*** from WordNet [160]. Specifically, we define two additional relations in $\mathcal{R}$. The first relation corresponds to the edges between a word and its synonyms (also called synsets) defined in WordNet. In particular, given a word w_i, we extract its synonyms, denoted by Syn_i. We define the set of synonym relation triples $SynT_i$ associated with w_i and its synonyms Syn_i as

$$SynT_i = \{(w_i, Synonym, w_n) | w_n \in Syn_i\} \tag{6.2}$$

with $Synonym \in \mathcal{R}$. We add $SynT_i$ for every word $w_i \in W$ to T.

The second relation corresponds to the edges between synonyms in the graph. Given a node w_{ni} which is a synonym of w_i, we extract the hypernyms of w_{ni}, denoted by Hyp_i. Then, we define the hypernym relation triples $HypT_i$ associated with w_{ni} and its hypernyms as

$$HypT_i = \{(w_{ni}, Hypernym, w_{nj})|w_{nj} \in Hyp_i\} \tag{6.3}$$

with $Hypernym \in \mathcal{R}$. T is expanded by adding all triples from $HypT_i$.

To have a complete directed graph, for all triples (s, p, o), we add the triples (o, p^{-1}, s) to T, with p^{-1} is the inverse of p. We have already added all cosine similarity edges because $hasCosine$ and $hasCosine^{-1}$ are identical. We calculate the triples from the inverse of $Synonym$ and $Hypernym$, denoted by $Synonym^{-1}$ and $Hypernym^{-1}$, respectively, and we add the calculated triples to T. We choose not to treat $Synonym$ and $Synonym^{-1}$ as identical relations because the domain and range have different types (word-WordNet entity edge).

Dataset-dependent knowledge: In order to incorporate dataset specific information, we connect table nodes with word nodes using TF-IDF relations. In particular, we calculate the TF-IDF value for a word w_i in table $t \in C$, denoted as TF-IDF(t, w_i). We follow the same binning approach that we used for discretizing semantic similarities in order to obtain a triple from TF-IDF(t, w_i). For example, given n_{tfidf} different intervals, if TF-IDF(t, w_i) belongs to the k-th interval, we obtain the triple $(t, hasTFIDF^k, w_i)$ which is added to T. In this case, we expand $\mathcal{R}$ by adding n_{tfidf} relations that are related to TF-IDF.

The cosine similarity between two words from a pretrained embedding encodes the co-occurrence information in the large pretraining corpus. By encoding the local co-occurrence information in our table collection, the constructed knowledge graph can retain dataset-specific relations. If we take the headers of tables as an example, in multiple tables, we could frequently find this sequence of headers (with different orders): Birth date, Birth place, Death date, Death place, etc. So the co-occurrence of tokens in this sequence of headers should be high. We utilize PMI to describe local context information using a sliding window strategy. The edge weight of each pair of words is calculated by:

$$PMI\left(w_i, w_j\right) = \log \frac{p\left(w_i, w_j\right)}{p\left(w_i\right) p\left(w_j\right)} \tag{6.4}$$

where $p\left(w_i, w_j\right)$ is the probability of co-occurrence of words w_i and w_j in the same sliding window of length s_w, which is estimated by:

$$p\left(w_i, w_j\right) = \frac{\#N_{\text{co-occurrence}}\left(w_i, w_j\right)}{\#N_{\text{windows}}} \tag{6.5}$$

with $\#N_{\text{co-occurrence}}\left(w_i, w_j\right)$ is the number of times the pair $\left(w_i, w_j\right)$ co-occurs in the same sliding windows over the whole table collection, and $\#N_{\text{windows}}$ is the total number of sliding windows of size s_w over the whole collection of tables.

Binning is used again for $PMI(w_i, w_j)$. Given n_{pmi} intervals for the set of PMI values, if $PMI\left(w_i, w_j\right)$ belongs to the k-th interval, we obtain the triple $\left(w_i, hasPMI^k, w_j\right)$. The triple $\left(w_j, hasPMI^k, w_i\right)$ is also valid because the PMI calculation is symmetric, and we add both triples to T. After adding the triples, we expand $\mathcal{R}$ by adding n_{pmi} relations that are related to PMI.

We add the inverse relations in order to finish constructing the graph. Like the $hasCosine$ relation, $hasPMI$ and $hasPMI^{-1}$ are identical. For TF-IDF, we define a new relation TF-IDF$^{-1} \in \mathcal{R}$ in order to compute the directed edges from words to table nodes.

6.2.2 Knowledge Graph Embedding Learning

To learn an embedding for each node in the graph (**RQ2**), we use link prediction as the pretraining task to learn node embeddings of the constructed knowledge graph in section 6.2.1. The objective of link prediction is to predict new facts given by (s, p, o) triples. So, the directed labeled graph G contains only a subset of possible edges, and the objective is to predict the score $f(s, p, o)$ of a possible edge (s, p, o) to determine the validity of the triple. We use the graph auto-encoder model introduced by Schlichtkrull et al. [208] that consists of a node encoder and scoring function for the decoder. The role of the encoder is to compute the embedding $e_i \in \mathbb{R}^d$ of node v_i in the graph. So, the encoder is the R-GCN model, and the node embedding is obtained by setting e_i to h_i^L, where L is the number of layers in R-GCN and h_i^L is the hidden representation of node v_i from the last layer. The role of the decoder is to reconstruct edges of the graph using node embeddings. This means that the decoder maps (s, p, o) triples into a real valued score. The advantage of the encoder-decoder architecture is the end-to-end training of both the embedding and the scoring function.

The decoder function is based on DistMult factorization [269] that has shown good results in link prediction despite its simple expression. Every relation $p \in \mathcal{R}$ is associated with a diagonal matrix $R_p \in \mathbb{R}^{d \times d}$, and the score $f(v_i, p, v_j)$ of triple (v_i, p, v_j) is given by:

$$f(v_i, p, v_j) = e_{v_i}^T R_p e_{v_j} \tag{6.6}$$

where e_{v_i} and e_{v_j} are the embeddings of nodes v_i and v_j respectively. The graph auto-encoder is trained with negative sampling as in [269, 208, 243]. In particular, we treat the triples in T as positive triples. For each $t_p \in T$, we sample w negative samples by either corrupting the object or subject of t_p. We optimize the graph auto-encoder parameters via link prediction by minimizing the cross-entropy loss:

$$\mathcal{L} = -\frac{1}{(1+\omega)|E|} \sum_{(s,p,o,y) \in \mathcal{T}} y \log \sigma(f(s,p,o)) + (1-y) \log(1 - \sigma(f(s,p,o))) \tag{6.7}$$

where $\mathcal{T}$ represents the set of positive and corrupted triples, y is the label of triple which is set to 1 for positive triples, and 0 for corrupted triples, and σ is the logistic sigmoid function. Minimizing the cross-entropy loss leads to having a higher $f(s, p, o)$ score for positive triples than the corrupted ones.

6.3 KG Embedding for Table Retrieval

Our proposed heterogeneous graph includes information from both the table collection and external resources, where various relations among nodes are encoded. As described in section 6.2.2, after training the graph auto-encoder on the link prediction task, the encoder provides an embedding for each node in the graph that captures the graph structure and the information that is passed from node to node using the edges labeled by relations from $\mathcal{R}$.

We show how to incorporate node representations into a learning-to-rank (LTR) model (**RQ3**) designed for table retrieval. Our trained graph auto-encoder simultaneously provides embeddings for different types of nodes (tables, words, synsets) in the knowledge graph so that we can tackle the table retrieval problem by using each embedding type independently or by having a joint model that combines the multiple embeddings. In this section, we discuss multiple LTR models that take advantage of our proposed graph embeddings in order to improve the results of ad hoc table retrieval. In training, a set of queries $Q = \{q_1, q_2, \ldots, q_s\}$, and a table corpus $\mathcal{C} = \{t_1, t_2, \ldots, t_l\}$, are given, where s and l are the total number of queries and data tables, respectively. We denote the number of tokens per query $q \in Q$ by n, and the number of tokens per table $t_j \in \mathcal{C}$ by m. Each table t_j has a relevance score, denoted by y_j, to a given query q. We propose f_w, a new joint embedding LTR model with parameters w, which incorporates multiple embedding spaces to predict the relevance score of a given query-table pair (q, t_j), such that higher ranked tables should be more relevant to the query.

6.3.1 Graph word embedding:

The first type of embedding used in our LTR model is the word embedding obtained from word nodes. After using the encoder to calculate the embedding of each node, we collect word nodes to form a word vocabulary for the tables collection, which is used to compute the word embeddings of queries and tables. For a given query $q = q_1, q_2, \ldots, q_m$ where m is the length of the query and q_l is the l-th token of q, the R-GCN word representation is given by

$$\boldsymbol{q} = \boldsymbol{q_1} \oplus \boldsymbol{q_2} \oplus \boldsymbol{q_3} \oplus \cdots \oplus \boldsymbol{q_m} \tag{6.8}$$

where $\boldsymbol{q_k} \in \mathbb{R}^d$ is a d-dimensional R-GCN word embedding of token q_k and $\oplus$ is the concatenation operator to build the matrix $\boldsymbol{q} \in \mathbb{R}^{m \times d}$. A given table t_j is linearized by concatenating metadata, such as table caption, page title, headers, and data values. Then, R-GCN word embeddings are used to map t_j to an embedding matrix $\boldsymbol{t_j} \in \mathbb{R}^{n \times d}$.

For a given query-table pair (q, t_j), the inputs to the word embedding-based LTR model are $\boldsymbol{q}$ and $\boldsymbol{t_j}$. We choose the Convolutional Kernel-based Neural Ranking Model (Conv-KNRM), proposed by Dai et al. [51], as our word embedding-based LTR architecture.

Conv-KNRM uses a Convolutional Neural Network (CNN) to embed n-grams of the query and document into a unified embedding space, and computes the similarity between each pair of n-gram embeddings. These similarities are compared to a set of K kernels, where each kernel is a normal distribution with a given mean and standard deviation. Then kernel-pooling [263] is used to summarize the similarities into a soft-matching feature vector of dimension K; intuitively, this vector represents the probabilities that the similarities come from the distribution specified by each kernel. The soft-matching feature vector is computed for different n-grams of query and document, and then they are concatenated into a single feature vector. The extracted feature is then passed through a learning-to-rank layer to predict a relevance score. We choose Conv-KNRM as the main component in our LTR model because it shows good performance in multiple benchmarks for document retrieval. Moreover, the CNN approach of modeling n-grams makes cross-matching between query and document tokens feasible, effective, and efficient. The input embedding layer to conv-KNRM is initialized using our word embeddings.

6.3.2 Graph embeddings for WordNet entities:

Given a query $q = q_1, q_2, \ldots, q_m$ where m is the length of the query and q_l is the l-th token of q, we translate each token q_l into a set of synonyms and hypernyms, denoted by $Trans(q_l)$, using WordNet. First, we extract the set of synonyms from WordNet and we append it to $Trans(q_l)$. Then, we use a stack to extract the hypernyms of synonyms, and then the transitive closure, with a maximum of 20 hops, over hypernyms. So, $Trans(q_l)$ forms a sequence of synonyms and hypernyms from WordNet. Finally, the translated query, $Trans(q)$, is given by:

$$Trans(q) = [Trans(q_1); Trans(q_2); \ldots; Trans(q_m)] \tag{6.9}$$

We apply the same idea to obtain the translation $Trans(t_j)$ of a given table t_j. After the translation step, our queries and tables are represented as a sequence of WordNet entities. Given that our graph auto-encoder produces embeddings for WordNet entities (synonyms and hypernyms), we compute WordNet embeddings for the translated query and table:

$$\boldsymbol{Trans(q)} = \bigoplus_{w_n \in Trans(q)} \boldsymbol{w_n} \; ; \boldsymbol{Trans(t_j)} = \bigoplus_{w_n \in Trans(t_j)} \boldsymbol{w_n}$$

where $\boldsymbol{w_n} \in \mathbb{R}^d$ is a d-dimensional R-GCN WordNet embedding of w_n, $\boldsymbol{Trans(q)} \in \mathbb{R}^{|trans(q)| \times d}$ and $\boldsymbol{Trans(t_j)} \in \mathbb{R}^{|trans(t_j)| \times d}$ are the R-GCN embedding matrices of $Trans(q)$ and $Trans(t_j)$, respectively. We train a Conv-KNRM model on table-query pairs of translated sequences using $\boldsymbol{Trans(q)}$ and $\boldsymbol{Trans(t_j)}$ as inputs, and with an embedding layer that is initialized using the R-GCN embedding for WordNet nodes.

6.3.3 Graph table embedding:

The embeddings for both word and WordNet nodes can be used directly for tables and query tokens. In contrast, embeddings of table nodes only provide representations for tables. For a given query-table pair (q, t_j), in order to compute an embedding for queries using table node embeddings, we propose an approach inspired by pseudo relevance feedback, where we generate a pseudo-query by aggregating the top-J tables returned by BM25. In particular, given a query q and a collection of tables $\mathcal{C}$, the pseudo-query q' is the sequence of closest J tables, $t_1, t_2, \ldots, t_J$, to q using BM25. The R-GCN table embeddings are used to compute the embedding matrix $\boldsymbol{q'} \in \mathbb{R}^{J \times d}$ which is given by

$$\boldsymbol{q'} = \overline{\boldsymbol{t_1}} \oplus \overline{\boldsymbol{t_2}} \oplus \overline{\boldsymbol{t_3}} \oplus \cdots \oplus \overline{\boldsymbol{t_J}} \tag{6.10}$$

where $\overline{\boldsymbol{t_i}}$ is the table embedding of t_i which is computed using R-GCN for table nodes in the knowledge graph.

Then, we aggregate $\boldsymbol{q'}$ using a simple aggregation function for neural networks, in order to compute the query embedding. Our neural aggregation function is based on ARC-I [95] which summarizes the meaning of a sequence through layers of convolution and pooling, and produces a fixed length feature vector $\boldsymbol{q_{agg}}$. In our case, the input sequence to ARC-I is $\boldsymbol{q'}$. The last layer of the feature extractor of the table embedding model consists of a pointwise multiplication layer between the table embedding $\overline{\boldsymbol{t_j}}$ of t_j and $\boldsymbol{q_{agg}}$. The resulting feature vector is passed through a multilayer perceptron (MLP) to predict the relevance score of (q, t_j).

6.3.4 Joint embedding:

We describe phase II of our proposed MultiEm-RGCN which combines all three types of embeddings into one LTR model. As shown in Figure 6.2, the phase I embeddings

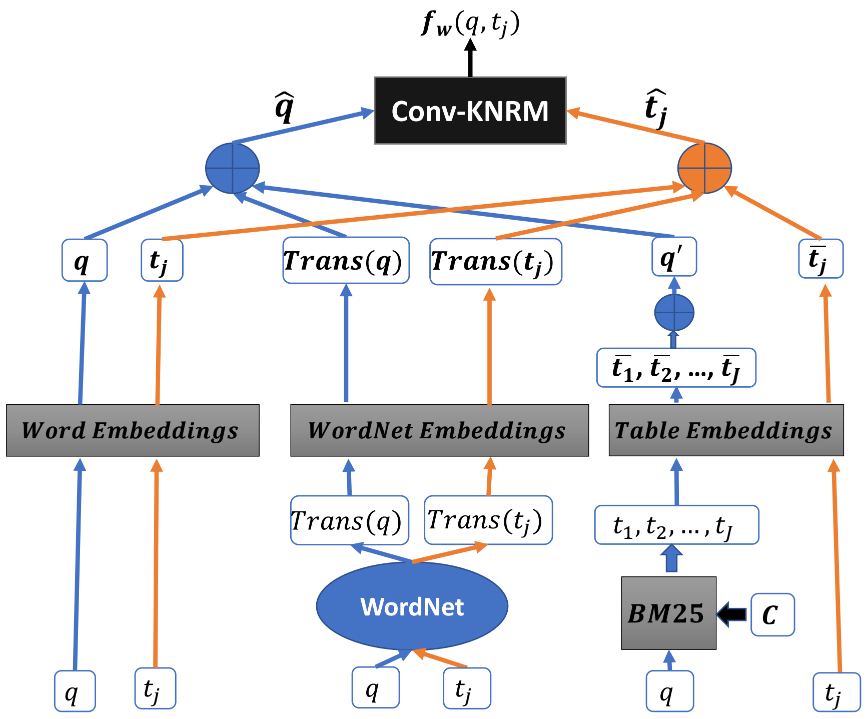


Figure 6.2: Overview of phrase II of the proposed method MultiEm-RGCN. The blue and orange edges represent the data flow for a given query q and table t_j, respectively. $\oplus$ denotes the concatenation operator used to form the table embedding of pseudo-query $\boldsymbol{q'}$, the final query representation $\hat{\boldsymbol{q}}$, and the final table representation $\hat{\boldsymbol{t_j}}$. The Conv-KNRM bloc takes $\hat{\boldsymbol{q}}$ and $\hat{\boldsymbol{t_j}}$ as inputs to predict the final relevance score $f_w(q, t_j)$.

are used to compute the query representation $\hat{\boldsymbol{q}}$ and table representation $\hat{\boldsymbol{t_j}}$:

$$\hat{\boldsymbol{q}} = \boldsymbol{q} \oplus \boldsymbol{Trans}(\boldsymbol{q}) \oplus \boldsymbol{q'}; \quad \hat{\boldsymbol{q}} \in \mathbb{R}^{(n+|Trans(q)|+J)\times d}$$

$$\hat{\boldsymbol{t_j}} = \boldsymbol{t_j} \oplus \boldsymbol{Trans}(\boldsymbol{t_j}) \oplus \overline{\boldsymbol{t_j}}; \quad \hat{\boldsymbol{t_j}} \in \mathbb{R}^{(m+|Trans(t_j)|+1)\times d}$$

Then, we pass $\hat{\boldsymbol{q}}$ and $\hat{\boldsymbol{t_j}}$ through a Conv-KNRM model to predict the final relevance score of (q, t_j). The model is trained to minimize the listwise loss function ListNet [21], and generate a ranked list of tables for each query that matches the ranking using the ground truth relevance scores. We choose not to update phase I embeddings when minimizing the listwise loss function to reduce model complexity, and focus the efforts of training on learning the CNN filters and MLP weights of Conv-KNRM.

6.4 Evaluation

6.4.1 Baselines

Unsupervised ranking approaches

A table is considered as a single field document by concatenating indexable fields. For example, in the WikiTables collection, we concatenate table caption, attributes, data rows, page title and section title. We compare our approach against a single-field ranking method which is based on BM25 to calculate a retrieval score. On the other hand, a data table can be considered as multi-field document, so we compare against a multi-field ranking method that is based on the pretrained Glove word embedding when calculating cosine similarity. MaxTable [241] similarity measure is used to calculate the score between query tokens and a given field in a table.

Supervised ranking approaches

We compare our method against state-of-the-art approaches for table retrieval: LTR and STR [293]. We also compare against various embeddings that are used as input to Conv-KNRM. The first set of embeddings are pretrained on large text corpuses and are Word2Vec [158], Glove [186] and fastText [12]. The second set of embeddings are pretrained on WikiTables which are TabVec [76], and MCON [241].

6.4.2 Experimental Setup

In all reported results, we choose not to update the embeddings when minimizing the loss function for table retrieval for two reasons: first we would like to directly compare the quality of embeddings that we obtain from multiple methods. Second, by freezing word embeddings, we reduce model complexity, and focus the efforts of training on only updating the parameters of LTR model.

Table 6.1 summarizes the parameters that are used in MultiEm-RGCN. In each training step of R-GCN, we randomly construct a connected subgraph of size S_z to make computations feasible. Inverse relations enable constructing a subgraph with multiple types of nodes. For example, without the inverse of TF-IDF relation, it is not possible to add a table node to the subgraph when the current node is of type word. Given that our graph contains more edges connecting words and WordNet

Table 6.1: Parameters values used in our proposed model

Model phases	Parameters	Values
Phase I	number of layers L	2
	M_{cos} threshold	0.5
	n_{cos},n_{tfidf},n_{pmi} and intervals	3
	PMI sliding window size s_w	20
	Table selection probability p	0.0001
	Size of subgraph Sz	50000
	Dimension of embedding d	100
	Negative samples w	10
	optimizer	Adam optimizer with $l_r = 0.001$
Phase II	Number of kernels K	5
	number of extracted tables J	100
	number of layers in ARC-I	2 with 50 CNN each
	n-grams in Conv-KNRM	unigram, bigram, and trigram
	number of CNN filters per n-gram in Conv-KNRM	128
	length of query n	6
	number of tokens m per table	80
	optimizer	Adam optimizer with $l_r = 0.001$

nodes, we force including table nodes in the subgraph with probability p instead of picking a random node.

We evaluate the performance of our proposed method and baselines on the table retrieval task using Normalized Discounted Cumulative Gain (NDCG) [107], Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP).

6.4.3 Evaluation using the Wikitables corpus

Ranking results

Table 6.2 shows the performance of different approaches on the WikiTables collection. We show that our proposed method MultiEm-RGCN outperforms the baselines for all evaluation metrics.

Among R-GCN embeddings, the word-based embedding has better retrieval results than WordNet and table embeddings as shown in Table 6.2. This can be explained by the fact that the graph contains many edges that have word nodes as the subject or object. So, updating word nodes is more frequent than updating WordNet and table nodes. In addition to that, unlike table nodes that have only input edges from word nodes using $hasTFIDF^{-1}$ relation, and WordNet nodes that have only inputs from other word and WordNet nodes, word nodes receive input messages from all three types of nodes in the graph using multiple relations.

Table 6.2 shows that using only R-GCN word embedding leads to better retrieval

Table 6.2: Table retrieval evaluation results using our proposed embedding and baselines for WikiTables dataset

Category	Method	NDCG@5	MAP	MRR
Unsupervised ranking	Single-field ranking	0.4511±0.032	0.477±0.030	0.516±0.038
	Multi-field ranking	0.499±0.026	0.492±0.029	0.523±0.027
	MCON [241]	0.515±0.018	0.519±0.021	0.532±0.019
Supervised ranking	LTR [18, 10]	0.514±0.039	0.522±0.035	0.570±0.019
	STR [293]	0.582±0.037	0.591±0.037	0.636±0.037
	Conv-KNRM+Glove [186]	0.595±0.033	0.598±0.032	0.629±0.031
	Conv-KNRM+fastText [12]	0.601±0.032	0.601±0.034	0.636±0.030
	Conv-KNRM+Word2vec [158]	0.600±0.031	0.600±0.025	0.631±0.026
	Conv-KNRM+TabVec [76]	0.595±0.036	0.602±0.034	0.635±0.036
	Conv-KNRM+MCON [241]	0.582±0.034	0.583±0.033	0.624±0.036
MultiEm-RGCN	R-GCN Word embedding	0.613±0.033	0.607±0.032	0.641±0.035
	R-GCN WordNet embedding	0.526±0.092	0.535±0.072	0.575±0.084
	R-GCN Table embedding	0.474±0.044	0.486±0.041	0.535±0.045
	MultiEm-RGCN without $\boldsymbol{q}'$	0.620±0.028	0.619±0.028	0.651±0.028
	MultiEm-RGCN	**0.624±0.027**	**0.624±0.026**	**0.656±0.024**

results than the baselines, but it is not the same case for WordNet and table embeddings which are more useful when used in the joint model MultiEm-RGCN. Most of the Conv-KNRM based baselines have better results than STR, the state-of-the-art method for table retrieval. Conv-KNRM+MCON performs worse, likely because it only computes word embeddings for attributes. Among the baselines, Conv-KNRM combined with fastText achieves higher performance for all evaluation metrics. The use of character-level n-grams in fastText allows word embeddings to be created even for terms that have not been seen before, and reduces the negative effect of out of vocabulary tokens on calculating the final relevance score of a query-table pair.

We explain the improvement in performance of our model compared to baselines by two facts. First, our proposed graph combines rich semantic and lexical general knowledge from Glove and WordNet with data specific knowledge. This leads R-GCN to learn node embeddings with a balance between general knowledge and table collection characteristics. Second, our heterogeneous graph provides multiple embeddings that can be incorporated into a single LTR architecture in order to aggregate matching signals between query and table in multiple spaces. This leads to more accurate calculation of the relevance score of a query-table pair.

Adding more features

We examine the effect of adding data values and STR features to the MultiEm-RGCN model. Table 6.3 shows table retrieval results using MultiEm-RGCN with different combinations of features. Since it can be computationally expensive to process all values from a table, we randomly select 50 string values from each table, and append the value tokens to description and attribute tokens in phase II of MultiEm-RGCN. As shown in Table 6.3, we obtain slight improvements in retrieval results when adding random values to description and attributes.

Table 6.3: Table retrieval performance using MultiEm-RGCN with different features for WikiTables dataset

Method	NDCG@5	MAP	MRR
Description+ attributes	0.6246±0.0277	0.6242±0.0267	0.6565±0.0241
Description+ attributes+values	0.6263±0.0252	0.6256±0.0287	0.6574±0.0339
Description+STR+ attributes+values	**0.6272±0.0225**	**0.6285±0.0235**	**0.6595±0.0258**

STR represents the set of features for query, table, and query-table pairs and semantic features from various spaces. We use precalculated STR features from [293]. We append STR features to word, WordNet, and tables feature vectors, and then train end-to-end the full system. Table 6.3 shows that adding the large number of STR features only leads to a slight improvement over using only table content and metadata. Thus, not only does MultiEm-RGCN unified knowledge graph exceed the performance of specialized LTR [18, 10] and STR [293] features, but it also almost entirely captures any useful signal present in those features. R-GCN word, WordNet, and table embeddings are directly used in a joint LTR architecture, and this leads to a significant improvement over the state-of-the-art STR table retrieval method.

6.4.4 Evaluation using the WebQueryTable corpus

We also conduct experiments on WebQueryTable [268]. We compare the performance of our method against unsupervised and supervised baselines, except for LTR/STR because these methods require a wide range of features that are not provided in the dataset. Similar to WikiTables, we obtain three spaces of embeddings, which supports the hypothesis that MultiEm-RGCN simultaneously learns multiple types of embeddings from our heterogeneous graph. For the WebQueryTable dataset, there is only one relevant table per query, so MRR is always equivalent to MAP (and thus MRR and MAP are shown in the same column in Table 6.4). Consistent with WikiTables, our results on WebQueryTable show that incorporating multiple embeddings

Table 6.4: Table retrieval results for WebQueryTable.

Method	NDCG@5	MRR/MAP
Single-field ranking	0.5560	0.5362
Multi-field ranking	0.5849	0.5631
Conv-KNRM+Glove [186]	0.6004	0.5825
Conv-KNRM+fastText [12]	0.6097	0.5878
Conv-KNRM+Word2vec [158]	0.6072	0.5859
Conv-KNRM+TabVec [76]	0.6059	0.5856
Conv-KNRM+MCON [241]	0.5996	0.5798
MultiEm-RGCN	**0.6438**	**0.6200**

from MultiEm-RGCN into Conv-KNRM improves the evaluation metrics of table retrieval. This supports the hypothesis that MultiEm-RGCN captures rich semantic and lexical general knowledge from Glove and WordNet with data-specific knowledge when learning the embeddings of nodes. Then, as in WikiTables, our LTR model in MultiEm-RGCN combines matching signals from word, WordNet, and table nodes, which gives the possibility for query and table to be matched in multiple spaces.

6.4.5 Embeddings visualization

We visualize the embeddings learned by MultiEm-RGCN. Figure 6.3 shows the t-SNE visualization of node embeddings from the second layer in R-GCN using the WikiTables collection. Figure 6.3 shows three different spaces from embeddings which are: word (red dots), WordNet (green dots), and table (blue dots). For each embedding space, we randomly zoom a region to show the embeddings of nodes of our proposed graph. For word embeddings, we can see that the words *mobile*, *telephone*, *phone*, *online*, *internet*, *web*, *website*, etc., are close to each other. The same interpretation is valid for WordNet embeddings where synsets *bridge.v.03*, *bridge.v.04*, *crossing.n.05*, *lake.n.03*, *bridge.v.01*, *metro.v.01*, *train.v.10*, etc., are mapped to the same region in the WordNet embedding space. Finally, for table embedding space, a selected region has the tables *table-1064-451*, *table-1064-381*, *table-1064-384*, *table-1064-402*, *table-1047-153*, *table-1047-143*, *table-0938-612*, etc., in the zoomed region after t-SNE visualization. All these tables are related to the 2012 Summer Olympics, and they show the list of world records in the Olympics for multiple sport events.

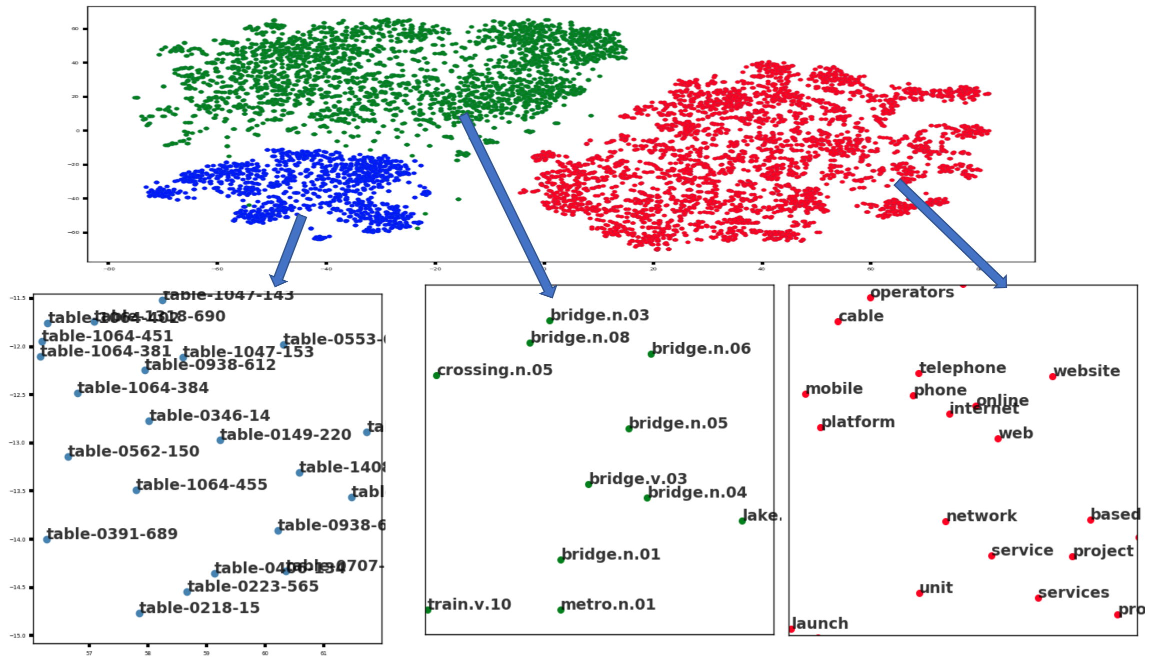


Figure 6.3: The t-SNE visualization of MultiEm-RGCN embeddings. There are three spaces of embeddings: word (red dots), WordNet (green dots), and table (blue dots). For each type of embedding, we zoom into a region to show the embeddings of nodes of our proposed graph.

6.5 Summary

In this chapter, we propose a novel table retrieval method denoted by MultiEm-RGCN. We have shown that a relational graph convolution network that incorporates both dataset-dependent knowledge and dataset-agnostic knowledge outperforms other pretrained embeddings on textual (Glove [185], fasText [12], Word2vec [158]) and table corpus (TabVec [76], MCON [241]). MultiEm-RGCN has two phases. The first phase consists of building a knowledge graph for a table corpus that contains multiple types of nodes and edges. This heterogeneous graph aims to capture data-agnostic knowledge that is semantic and lexical, and dataset-dependent knowledge that is derived from contextual information and term frequencies. A simple graph encoder with two R-GCN layers, and the DistMult decoder function are used to learn node embeddings by minimizing a link prediction loss function. The second phase consists of using R-GCN embeddings for the table retrieval task. This is achieved by building a new LTR model that combines word, WordNet, and table embeddings.

As a future research direction, it is possible to enrich the heterogeneous graph by data values signals. Data values represent the main content of datasets, and in many cases where the metadata is missing or inconsistent, data values are considered as the primary field that is used to rank datasets against the input queries. Therefore, gaining more understanding about data values is important. Our proposed heterogeneous graph captures semantic and lexical signals from external resources. To better understand data values, it is possible to exploit the semantics of an existing knowledge base (KB). For example, the information from KB can be incorporated into our graph by linking the dataset content (headers and data values) to KB components such as entities, classes, and properties. It is possible to use an RDF-based KB which is composed of terminologies (TBox) and assertions (ABox). The TBox is composed of the RDF Schema (RDFS) definitions of classes, class relations (*rdfs:subClassOf*), properties (*rdfs:domain*, *rdfs:range*, *rdfs:subPropertyOf*). The ABox is composed of entities in the form of RDF triples <*subject, predicate, object*>, where *subject* represents an entity, *predicate* represents a property, and *object* can be an entity or data value. The class of a given entity is defined using *rdf:type.* Semantic reasoning over KB can capture implicit knowledge from RDF triples. For example, given an entity a, two classes c_1 and c_2, and the triples <a, *rdf:type,* c_1> and <c_1, *rdfs:subClassOf,* c_2>, it is possible to infer that <a, *rdf:type,* c_2>. These semantics from existing KB provide additional ways to better understand data values in table collections from a dataset-agnostic knowledge perspective.

The row and column dependencies between tokens in a table can be lost by either computing PMI on the flattened table when learning the graph embedding in phase I, or concatenating tokens of a table when using the graph embedding for table retrieval in phase II. In the next chapter, we propose a new learning-to-rank method for table retrieval that takes into account the row and column dependencies between tokens so that the structural information of a data table is incorporated into the neural architecture.

Chapter 7

Deep Semantic and Relevance Matching Model for Learning to Rank Data Tables

7.1 Introduction

Supervised learning, based on features from tables, queries, and query-table pairs [18, 10], has resulted in the best performing table retrieval systems. Building on this, Zhang and Balog [293] proposed extending these features with semantic matching between queries and tables using various semantic spaces which are: Word embeddings, Graph embeddings, Bag-of-entities and Bag-of-categories. Semantic and traditional features are then used to train a supervised model called STR [293]. However, there are major drawbacks of STR for ad hoc table retrieval. First, they are based on hand-crafted features, and that limits the ability to capture multiple levels of similarity between query and table. Second, they ignore query relevance matching which is an important matching signal in document retrieval in general. Third, they assume equal contribution of each query token to the final relevance score when ranking web tables against a given query.

Multiple matching signals, such as semantic and lexical signals, are captured by representing data tables as an heterogeneous graph in MultiEm-RGCN [238]. However, the graph representation in MultiEm-RGCN [238] ignores the structural information of data tables because the row and column dependencies between tokens in a table is neglected by either computing PMI on the flattened table when learning the graph embedding in phase I, or concatenating tokens of a table when using the graph embedding for table retrieval in phase II.

In order to overcome the limitations of prior methods in table retrieval, we propose a new model that combines deep contextual features with features based on term similarity distributions. Our model learns convolutional filters that extract contextual features from query/table interactions (semantic matching). This is combined with a feature vector based on the distributions of term similarity between queries and tables (relevance matching). Additionally, we incorporate table values into our model using row and column summaries that form the structural information. Finally, we learn the contribution of each query token to the final relevance score. These models are trained using a learning-to-rank approach with a listwise loss function. We show that our new method can improve table retrieval performance using a collection of tables from Wikipedia [293] and Web tables from a Microsoft dataset [268].

In summary, we make the following contributions:

- We propose a new semantic similarity model that is able to capture multiple levels of semantic signals between query and table. In order to capture contextual information, we apply various-sized convolutional filters to an interaction matrix built from the embeddings of query and table tokens, and then apply a second layer of convolutional filters to extract higher level features. Our representation of the table includes summary vectors about the contents of the table, both in terms of values in each column and values in selected rows.

- We demonstrate the usefulness of query relevance-specific components for the table retrieval task. Using kernel pooling, we learn a feature vector based on the probability distribution of the similarity of each document token to each query token, and we learn the contribution of each token to the final relevance score using a Term Gating Network. Each of these components lead to improvement on retrieval tasks without leading to a large increase in the number of parameters of the model.

- We compare our proposed method not only against methods from table retrieval and document retrieval, but we also adapt architectures from multiple domains to the table retrieval task. We show that ad hoc table retrieval benefits from table-specific architectures; that is, straightforward application of leading document retrieval approaches results in reduced performance.

7.2 Learning to Rank Data Tables

In this section, we introduce our proposed deep relevance model to enhance ranking of tables given a user's query. In training, a set of queries $Q = \{q^1, q^2, \ldots, q^m\}$ is given. Each query q^i is associated with a list of tables $t^i = (t^i_1, t^i_2, \ldots, t^i_{n^i})$, where t^i_j denotes the j^{th} table and n^i is the size of t^i. Each list of tables t^i is associated with a list of relevance scores $y^i = (y^i_1, y^i_2, \ldots, y^i_{n^i})$ where y^i_j denotes the relevance score of table t^i_j with respect to query q^i. We propose f_w, a new deep relevance model with parameters w, that is used to predict the relevance score of a given query-table pair (q^i, t^i_j). Our proposed model $f_w = M \circ F$ contains a feature extractor function F and a ranking model M. A feature vector $x^i_j = F(q^i, t^i_j)$ is created from each query-table pair (q^i, t^i_j). Then a ranking function M is used to predict a relevance score $M(x^i_j)$. So for a given query q^i and a list of tables t^i associated with the query, the objective is to obtain a list of scores $z^i = (M(x^i_1), M(x^i_2), \ldots, M(x^i_{n^i}))$. The predicted relevance scores are used to rank query-table pairs so that higher ranked tables should be more relevant to the query.

7.2.1 Listwise loss function for table retrieval

We propose using a listwise based loss function for table retrieval rather than relying on pointwise and pairwise approaches for two reasons. First, we are interested in training our model to generate a ranked list of tables for a given query without requiring the predictions of our model to match the ground truth relevance scores. Second, although negative sampling can be used in the pairwise approach to avoid the quadratic increase of query-table pairs, the pairwise strategy can increase data imbalance when there is a dominating class [144]. So the loss function L is given by:

$$L(w) = \sum_{i=1}^{m} l(y^i, z^i) \tag{7.1}$$

where l is a listwise loss function. We adapt the loss function proposed by Cao et al. [21] to the table retrieval task. Given a query q^i associated with a list of tables

t^i and relevance scores y^i, the feature extractor F extracts features from a given query-table pair, then a ranking model M generates a score list z^i. As in Cao et al. [21], the ground truth relevance scores and the predicted scores are converted into probabilities $P_{y^i}(q^i, t^i_j)$ and $P_{f_w}(q^i, t^i_j)$, respectively, using the softmax function. With cross entropy loss, for a query q^i, $l(y^i, z^i)$ equals to:

$$l(y^i, z^i(w)) = -\sum_{j=1}^{n^i} P_{y^i}(q^i, t^i_j) \times log(P_{f_w}(q^i, t^i_j)) \tag{7.2}$$

The gradient of $l(y^i, z^i(w))$ with respect to model parameters w is detailed in Cao et al. [21].

7.2.2 Deep relevance model architecture

We propose a new interaction-based deep semantic and relevance matching model (DSRMM) for table retrieval. There are two classes of neural architectures for ad hoc retrieval. Semantic similarity architectures treat the query and target as equals, and try to match them. Query relevance architectures exploit characteristics of the ad hoc retrieval task. Our hybrid model combines both concepts into one architecture. We extract semantic and relevance feature vectors from the deep semantic similarity model and query relevance matching network, respectively. The two features are then concatenated and passed through a fully connected layer to predict a retrieval score under the semantic and relevance settings. For a given query q^i and table t^i_j, the final relevance score is given by

$$f_w(q^i, t^i_j) = NN_c([SS(q^i, t^i_j); QR(q^i, t^i_j)]) \tag{7.3}$$

where SS is the semantic similarity neural network, QR is the query relevance neural network, and NN_c is a neural network used to predict the relevance score from a vector concatenating the outputs of the semantic and relevance networks.

Inputs to Networks

The input to our architecture is a query-table pair. A given table t^i_j contains description, cell values, and attributes or headers as shown in Figure 7.1. The description denotes the metadata of the table such as page title, section title, table caption, etc.

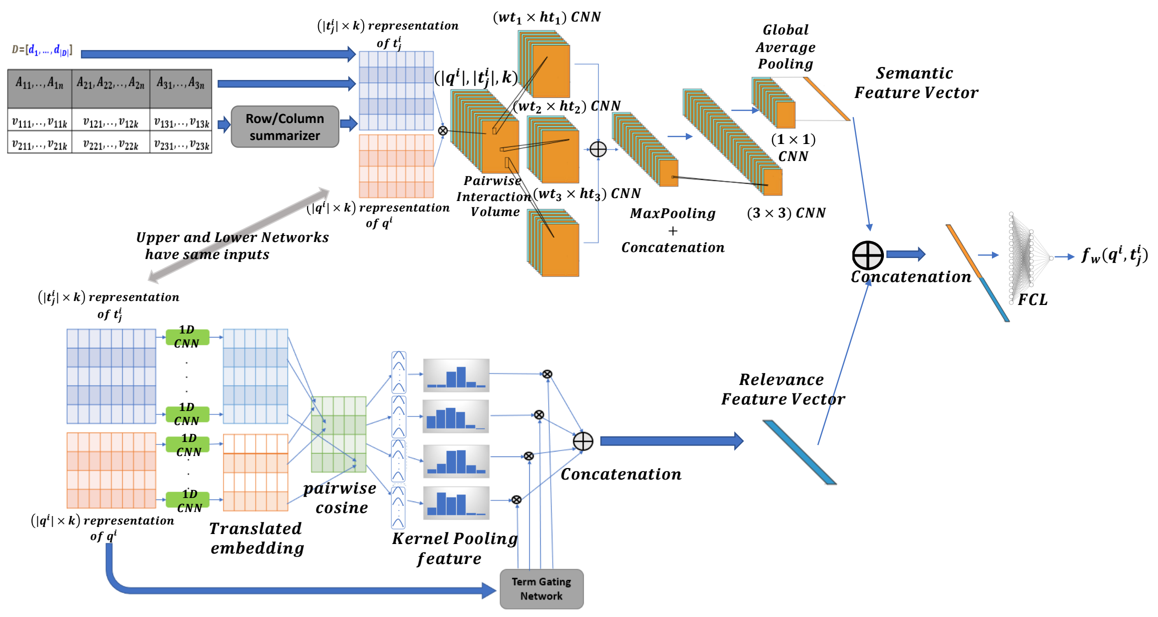


Figure 7.1: Architecture of Deep Semantic and Relevance Matching Model (DSRMM) for table retrieval. $\oplus$ denotes the concatenation operator, and $\otimes$ is the pointwise multiplication operator. The semantic and relevance networks extract semantic and relevance feature vectors respectively. The concatenated vector of semantic and relevance features is passed through a fully connected network to predict the final relevance score between the query and table.

The input representation of a table, denoted by T_j^i, contains the pretrained Glove word embeddings of description and attributes. Cell values contain rich information that can be used to match query and tables. Some queries depend on the presence of specific columns, others depend on the presence of specific rows. In order to incorporate row and column representations into T_j^i, we present a row/column summarizer component that compresses each row and each column into a fixed length feature vector. In particular, given the k-th row r_k and l-th column c_l from t_j^i, the outputs of the summarizer component $S(r_k)$ and $S(c_l)$ are given by:

$$S(r_k) = \frac{1}{|r_k|}\sum_{w\in r_k} v_w \quad \text{and} \quad S(c_l) = \frac{1}{|c_l|}\sum_{w\in c_l} v_w$$

where v_w is the word embedding of token w, and $|r_k|$ and $|c_l|$ are the number of tokens in the k-th row and l-th column of t_j^i, respectively. A table t_j^i with n_r rows and n_c columns results in $n_r + n_c$ additional feature vectors that are concatenated to T_j^i to form the $|t_j^i| \times k$ table representation, where k is the dimensionality of word embeddings. The query representation, denoted by Q^i, of q^i is calculated using the word embedding of each token in the query. So the dimensionality of Q^i is $|q^i| \times k$, where $|q^i|$ is the length of the query. For the rest of the chapter, we assume that $|t_j^i|$ is equal to m for all tables, and $|q^i|$ is equal to n for all queries.

Semantic similarity component

Semantic similarity extracts contextual features from query-table interactions to infer the semantic meaning and relation between query and table. The top portion of Figure 7.1 illustrates the semantic similarity component, SM, in detail. To capture semantic similarity, we build the interaction matrix X between query and table using the pointwise multiplication between pairwise rows from query representation Q^i and table representation T_j^i. In tables, token order only matters locally; the rows and columns could be arbitrarily ordered without changing the meaning of the table. Thus, there is less utility in encoding sequences with the bi-LSTM models as is done in Match-Tensor [104].

To capture multiple levels of similarity between the query and table, we propose using multiple convolutional filters with different width and height. The width indicates the number of query tokens that are used in the convolution. The set of width values is given by $\{wt_1, wt_2, wt_3\}$. The height value indicates the number of tokens from the table that are used in the convolution. The set of height values is given by $\{ht_1, ht_2, ht_3\}$. Each table is represented as a matrix $T_j^i \in \mathrm{I\!R}^{\mathrm{m \times k}}$, and each query is represented as a matrix $Q^i \in \mathrm{I\!R}^{\mathrm{n \times k}}$. After pointwise multiplication of each query-table pair of tokens, we obtain the interaction tensor X with dimension $n \times m \times k$. We pass X through k_1 filters of size $(wt_1 \times ht_1)$, k_2 filters of size $(wt_2 \times ht_2)$, and k_3 filters of size $(wt_3 \times ht_3)$ to obtain $X_{wt_1 \times ht_1}$, $X_{wt_2 \times ht_2}$, and $X_{wt_3 \times ht_3}$ feature maps respectively. We apply a max pooling operation to each feature map, and we concatenate the resulting tensors into one tensor with size $(n/2 \times m/2 \times (k_1 + k_2 + k_3))$. In order to extract high level semantic interactions between the table and query, we apply k_4 convolutional filters of size (3×3); then we reduce the number of depth channels using s filters with size (1×1). We summarize the information obtained in each channel using Global Average Pooling (GAP) to obtain our semantic similarity feature vector $SS(q^i, t_j^i)$.

Query relevance component

Guo et al. [80] demonstrated that many neural models for document retrieval depended on semantic similarity, but this is an inappropriate measure for the task. They argued that ad hoc retrieval depends on exact matching and query term importance, and that sometimes queries only need to match part of the document, not the document as a whole. Semantic similarity on the other hand assumes that the items being matched are roughly equivalent in scope, that their meanings are composed from their parts, and that items must be matched in their entirety. We believe that this argument for a specialized architecture applies equally to table retrieval. However, because semantic similarity still provides a useful signal based on context, we design a model that includes the best features of both approaches.

Our query relevance component adapts Guo et al.'s matching histogram mapping [80] to the table retrieval task. The matching histogram mapping is based on a hard assignment of matching similarities between a given query token and the table tokens. This histogram-based feature counts the number of table tokens whose similarity to the query token is within the bin's range. However, this representation is not differentiable and not computationally efficient. Therefore, we adapt kernel-pooling [263] for soft-match signals to the table retrieval task. The objective of using kernel pooling is to extract a soft matching histogram between a given query token and table tokens. Given a query token q_l^i and table t_j^i, we use $r1$ 1-D convolutions to translate each token. Then we calculate cosine similarity between the translated tokens. There are two advantages of the convolution. First, it allows us to learn similarities that are present in our query/table collection but that were not captured by the Glove [186] corpus. Second, instead of updating the word embedding ($|V| \times k$ parameters, where V is the vocabulary), we update the convolutional filters ($r1 \times k$ parameters), so that we decrease the complexity of the model (as $r1 << |V|$).

q_l^i is embedded to $\mathbf{q_l^i}$, then translated to $\mathbf{v_l^i}$. The cosine similarity between $\mathbf{v_l^i}$ and the s^{th} token of t_j^i is given by C_{ls}. Suppose that we have K kernels for soft matching, with mean $\mu = \{\mu_1, \mu_2, \dots, \mu_K\}$, and standard deviation $\sigma = \{\sigma_1, \sigma_2, \dots, \sigma_K\}$; the soft matching assignment of query token q_l^i to k^{th} kernel is given by

$$K_k(q_l^i, t_j^i) = \sum_{s=1}^{m} exp(-\frac{(C_{ls} - \mu_k)^2}{2\sigma_k^2}) \tag{7.4}$$

K_k calculates the soft matching similarities around μ_k with a variance σ_k^2. The closer C_{lm} is to μ_k, the higher is K_k. The kernel pooling feature vector of query token q_l^i and table t_j^i is given by:

$$KP(q_l^i, t_j^i) = [K_1(q_l^i, t_j^i); K_2(q_l^i, t_j^i), \ldots, K_K(q_l^i, t_j^i)] \tag{7.5}$$

Since exact matching is an important signal in any retrieval task, we reserve the first kernel K_1 for soft exact matching. So, we set μ_1 to 1, and σ_1 to 0.001.

Query tokens are not equally important for relevance matching. In order to model each query token's importance, we use a Term Gating Network (TGN) [80] to control the contribution of each query token to the final relevance score. For a given query q^i, the gating function is given by:

$$g_j = \frac{exp(\mathbf{w_g q_j^i})}{\sum_{l=1}^{n} exp(\mathbf{w_g q_l^i})} \tag{7.6}$$

where $\mathbf{w_g}$ is the weight vector of the term gating network, and $\mathbf{q_l^i}$ is the embedding of query token q_l^i, The final feature vector using relevance matching is given by:

$$QR(q^i, t_j^i) = [h_1^{ij}; h_2^{ij}, \ldots, h_{|q^i|}^{ij}] \tag{7.7}$$

with

$$h_l^{ij} = g_l \times KP(q_l^i, t_j^i) \tag{7.8}$$

Our final model captures both semantic similarity and query relevance matching which both play an important role in ad hoc table retrieval.

7.3 Evaluation

We evaluate our approach using two different data collections and compare it against a number of baselines.

7.3.1 Baselines

We compare the performance of our proposed model against several baselines from multiple fields.

Unsupervised table retrieval

We compare the performance of our proposed method against MCON [241] which is based on new word embeddings for attribute tokens of tables. When calculating the retrieval score of MCON, we use the MaxTable ranking method which was shown to be the best ranking method for unsupervised table retrieval [241]. MaxTable is a late fusion similarity model that finds the closest table token to each query token using cosine similarity, and then sums over these similarities.

Ad-hoc table retrieval methods

We compare our method against state-of-the-art approaches for table retrieval: LTR [293], STR [293], and MultiEm-RGCN [238]. Table2Vec [290] is another system that solves the same task, but the reported performance is lower than that of STR. Therefore, we do not include Table2Vec in our evaluation.

Unsupervised document ranking approaches

Unsupervised document ranking approaches can be applied to table retrieval if a linearization is applied to the table to create a sequence of terms. Depending on the structure of the table, we obtain two categories of baselines:

Single-field document ranking: A table is considered as a single field document by concatenating indexable fields. For example, in the WikiTables collection, we concatenate: table caption, attributes, data rows, page title and section title. We compare our approach against three baselines in the category of single-field document ranking. The first single-field ranking method, SingleField-BM25, is based on BM25 to calculate a retrieval score. In the second ranking method, called SingleField-LM, we estimate a language model [45, 283] for the formed document in order to rank a given table against the query. Finally, for the third approach, called SingleField-P, we calculate the score of a query-table pair using word embedding-based ranking method MaxTable [241]. This is different from MCON in that it uses pre-trained Glove embeddings and does not generate predicted contexts.

Multi-field document ranking: In a multi-field ranking scenario, a table is defined using multiple fields. For example, in WikiTables data, the fields are: page title, section title, table caption, attributes and table body or values. We compare our method against two baselines in the category of multi-field document ranking.

The first multi-field ranking method, MultiField-LM, is based on combining language models for multi-field documents. The second multi-field ranking method, MultiField-P, is based on the pretrained Glove word embedding when calculating cosine similarity. MaxTable [241] similarity measure is used to calculate the score between query tokens and a given field in a table.

Learning to rank for document retrieval methods

We compare our proposed method against C-DSSM [214], ARC-I [95], ARC-II [95], DUET [161], Match-Tensor [104], DRMM [80], and Conv-KNRM [51]. A given document is the concatenation of the description and attributes of a table.

Sentence classification

Kim [117] proposed a CNN for sentence classification. A 1-D convolutional layer with multiple filter widths is applied to the concatenation of word embeddings of sequence tokens. Each filter produces a feature map of a different size. A max-pooling operation is then applied over each feature map to take the maximum value, and this is finally fed into a fully-connected layer. In our table retrieval task, a given sentence is the concatenation of tokens in description, query, and attributes. The final output is a relevance score instead of a classification score.

Document classification

HAN [274] is a hierarchical attention network for document classification. It is composed of four parts: a word sequence encoder, a word-level attention layer, a sentence encoder and a sentence-level attention layer. A GRU-based [39] sequence encoder is used to encode each token in a given sentence into a hidden state. Then an attention mechanism is applied on the word level to extract the most important words to the meaning of the sentence, and aggregate the representation of those informative words to produce a single feature vector related to the sentence. A second GRU-based sequence encoder is used to encode sequences; then a sentence-level attention mechanism is applied to obtain a document vector that summarizes all the information of sentences in a document. In a table retrieval scenario, a document contains sentences from the query, table description, and attributes. We predict a relevance score for each document formed from a query-table pair using HAN architecture.

Table type classification

Nishida et al. [171] proposed a hybrid deep neural network architecture, called TabNet, for table classification. The architecture is composed of a recurrent neural network (RNN) to encode a sequence of tokens of each cell. The next component in the architecture is the attention mechanism from Yang et al. [274] which extracts important tokens from each cell, and forms an input volume. The constructed volume is passed through a Convolutional Neural Network (CNN), and then a fully connected layer is added to compute the output layer. To use TabNet for table retrieval, we add a cell to the table that contains the query. So, in addition to encoding the original cells of a given table using RNN, we also encode query tokens. Then we apply CNN to the constructed volume.

7.3.2 Experimental Setup

In our proposed model, we use an existing pretrained neural word embedding from Glove with $k = 300$. We choose not to update the word embedding when minimizing the listwise loss function for two reasons: first we have far fewer labeled query-table pairs than examples from the unlabeled text corpus used to train the Glove model. Second, by freezing word embeddings, we reduce model complexity, and focus the efforts of training on extracting semantic and relevance matching.

We train our model for 30 epochs, and each batch contains only tables that are candidates of a given query in order to calculate the listwise loss. We use Adam optimizer [120] for gradient descent to minimize the loss function, and update the weights of our model. We set the learning rate to 0.0001. The model is implemented using PyTorch, with Nvidia GeForce GTX 1080. We set the number of query tokens n to 6, and the number of table tokens m to 100. The m table tokens contain the first 50 tokens from description, first 30 tokens from attributes, and 20 rows and columns. We start by including column summaries, and then row summaries because in many cases, tables contain more rows than columns. We set the number of CNN filters of the first layer $k1$, $k2$, and $k3$ to 20. The set of width values $\{wt_1, wt_2, wt_3\}$ is equal to $\{3, 5, 7\}$, and the set of height values is equal to $\{3, 3, 3\}$. So we use 3 query tokens in each convolution. $k4$, which is the number of CNN filters in the second layer, is equal to 200. The dimensionality of the feature vector s of semantic matching component is 100. We set the number of kernels in the relevance matching component to 5. So, given that the cosine similarity is between $[-1, 1]$, the means of the kernels are $\mu = [1, 0.75, 0.25, -0.25, -0.75]$. The standard deviations of the kernels are $\sigma = [0.001, 0.1, 0.1, 0.1, 0.1]$. We reserve the first kernel ($\mu_1 = 1$ and $\sigma_1 = 0.001$) for exact match because it is an important signal in retrieval tasks.

In order to calculate the retrieval score for query-table pairs by combining language models, we use the implementation in Hasibi et al. [88] which is based on Elasticsearch.

7.3.3 Experimental results

We evaluate the performance of our proposed method and baselines on the table retrieval task using Normalized Discounted Cumulative Gain (NDCG), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). All evaluation metrics results are reported using the TREC evaluation software, trec_eval[1].

WikiTables results

Table 7.1 shows the performance of different approaches on the WikiTables collection. We show that our proposed method DSRMM outperforms the baselines for all evaluation metrics. Consistent with what has been shown in ad hoc document retrieval, supervised approaches perform better on ad hoc table retrieval than unsupervised approaches.

Table 7.1: Table retrieval evaluation results for WikiTables dataset

Category	**Method**	**NDCG@5**	**MRR**	**MAP**
Unsupervised table retrieval	MCON [241]	0.515±0.018	0.532±0.019	0.519±0.021
Ad-hoc table retrieval	LTR [293]	0.514±0.039	0.570±0.038	0.522±0.035
	STR [293]	0.582±0.037	0.636±0.037	0.591±0.035
	MultiEm-RGCN [238]	0.625±0.027	0.657±0.024	0.625±0.026
Unsupervised document ranking	SingleField-BM25	0.451±0.032	0.516±0.038	0.477±0.030
	SingleField-LM	0.435±0.031	0.482±0.035	0.454±0.027
	SingleField-P	0.471±0.023	0.501±0.027	0.482±0.031
	MultiField-LM	0.464±0.037	0.492±0.035	0.475±0.038
	MultiField-P	0.499±0.026	0.523±0.027	0.492±0.029
LTR for document retrieval	C-DSSM [214]	0.510±0.027	0.548±0.026	0.521±0.026
	ARC-I [95]	0.553±0.033	0.607±0.032	0.553±0.029
	ARC-II [95]	0.567±0.029	0.613±0.029	0.562±0.029
	DUET [161]	0.524±0.037	0.579±0.041	0.528±0.035
	Match-Tensor [104]	0.569±0.030	0.613±0.034	0.565±0.031
	DRMM [80]	0.482±0.023	0.522±0.028	0.491±0.025
	Conv-KNRM [51]	0.595±0.033	0.638±0.031	0.608±0.032
Sentence classification	Kim [117]	0.566±0.034	0.617±0.035	0.564±0.037
Document classification	HAN [274]	0.567±0.034	0.614±0.037	0.565±0.033
Table type classification	TabNet [171]	0.570±0.030	0.616±0.029	0.568±0.029
Ad-hoc table retrieval	Our proposed model (DSRMM)	**0.640±0.029**	**0.680±0.028**	**0.642±0.029**

ARC-II outperforms ARC-I by directly learning from the interaction matrix, and

[1]https://trec.nist.gov/trec_eval/trec_eval.8.1.tar.gz

that enables early learning from query and table interactions rather than training a separate representation for each field. Compared to ARC-I, ARC-II combines the hierarchical modeling of individual tokens of query and table, and the patterns of their matchings using interaction matrix.

Among unsupervised table retrieval and document ranking approaches, MCON achieves higher performance for all evaluation metrics. This can be explained by the use of a mixed ranking model that incorporates the metadata of a table and the additional contexts in order to calculate the retrieval score.

Although Kim [117], HAN, and TabNet are not designed for table retrieval, we show that these architectures have competitive results compared to many of the other methods. Among these methods, TabNet has the best performance, and this can be explained by the fact that TabNet is designed for table type classification, so the input of TabNet is similar to table retrieval based methods. On the other hand, the primary input of HAN and Kim [117] are documents and sentences, respectively.

The deep semantic matching component of DSRMM extracts interactions between query tokens and table tokens using convolutional filters. We explain the improvement in performance of our model compared to baselines by three facts. First, the different low level filters capture multiple levels of similarity which are important in a retrieval task. Second, some patterns are hard to capture using only one layer of convolutional filters, so in DSRMM, the second set of convolutional filters identify high level interactions. Third, the semantic component is a position dependent component, and treats all query tokens equally. To solve that, our proposed relevance matching component provides position free and strength preserving histograms that are weighted by the importance of each query using a Term Gating Network.

To test significance, we use a two-tailed paired t-test between DSRMM and the best baseline reported in Table 7.1 which is Conv-KNRM. We found a t-test significance at level 0.05 for all metrics.

WebQueryTable results

For the WebQueryTable dataset, we compare the performance of our method against the most competitive methods found when using the WikiTables dataset: Kim [117], HAN, and TabNet. We also compare against the top two document retrieval approaches from Table 7.1: Conv-KNRM and Match-Tensor. We note that we do not compare to LTR/STR because these methods require a wide range of features that are not provided in the dataset. As shown in Table 7.2, our method outperforms Match-Tensor, Kim [117], HAN, and TabNet by a large margin. As with WikiTables, Conv-KNRM is the closest competitor. Conv-KNRM captures semantic matching of unigrams, bigrams, and trigrams between query and table tokens, but the query tokens are treated uniformly. For this dataset, there is only one relevant table per query, so MRR is always equivalent to MAP (and thus is not repeated in the table).

Table 7.2: Table retrieval evaluation results for WebQueryTable dataset. Here there is only one relevant table per query, so MRR is always equivalent to MAP.

Method	**NDCG@5**	**MRR/MAP**
Match-Tensor [104]	0.3232	0.3256
MultiEm-RGCN [238]	0.6232	0.6088
Conv-KNRM [51]	0.6052	0.5978
Kim [117]	0.3078	0.3097
HAN [274]	0.4620	0.4384
TabNet [171]	0.4876	0.4597
DSRMM	**0.6516**	**0.6345**

Analysis of alternative design choices

In order to justify the importance of each component in our proposed method, we present an ablation study of our hybrid model using the WikiTables dataset in Table 7.3. We train two versions of our model: the first version is only the semantic similarity component, and the second version is only the query relevance component. Our study shows that the semantic similarity network has better performance than the query relevance network. The full architecture outperforms both isolated components, and adding query relevance to semantic similarity increases NDCG@5 from 0.601 to 0.640. Furthermore, removing the term gating network drops the performance of the query relevance component from 0.542 to 0.532, and this supports the idea of having different contributions to the relevance score for each query term.

We study the effect of removing the Row/Column summarizer from DSRMM as shown in the last row of the ablation analysis. So we define a baseline in which we randomly select 50 string values from each table, and append the values tokens to description and attributes tokens. Adding all values is computationally expensive and

Table 7.3: Analysis of alternative design choices using WikiTables dataset

Analysis	**Method**	**NDCG@5**	**MRR**	**MAP**
Ablation	Semantic similarity component only	0.601±0.031	0.640±0.033	0.596±0.031
	Query relevance component only	0.542±0.036	0.594±0.030	0.549±0.027
	Query relevance component only without TGN	0.532±0.034	0.583±0.030	0.537±0.032
	Query relevance only with 2 kernels	0.502±0.034	0.555±0.036	0.511±0.034
	DSRMM without Row/Column Summarizer	0.627±0.026	0.668±0.028	0.629±0.028
System variations	DSRMM+Pointwise loss	0.617±0.026	0.655±0.028	0.612±0.028
	DSRMM+STR	**0.642±0.021**	0.679±0.023	0.641±0.021
Proposed method	DSRMM	0.640±0.029	**0.680±0.028**	**0.642±0.029**

consumes a significant amount of memory. Removing the Row/Column summarizer results in a decrease for all evaluation metrics. Thus, each component of our system has a positive effect on the final results.

Table 7.3 shows a decrease in retrieval metrics for the query relevance component when using 2 kernels in kernel pooling as opposed to 5 kernels in original DSRMM. So, two kernels are not enough to extract fine-grained relevance matching for query-table pairs.

For system variations analysis, Table 7.3 shows that the listwise based approach leads to better retrieval results than the pointwise loss function which is consistent with what has been shown in document retrieval results. With listwise loss, DSRMM focuses on ranking the tables, rather than predicting the exact relevance score.

We study a second system variation that consists of adding STR features to the DSRMM model. STR represents the set of features for query, table, and query-table pairs and semantic features from various spaces. We use precalculated STR features from [293]. We append STR features to our proposed semantic and relevance features, and train our model. Table 7.3 shows that adding STR features leads to a slight improvement over vanilla DSRMM for NDCG@5. However, the DSRMM model trained with description, attributes, and row and column summaries has the best performance for MRR and MAP, by using only word embedding space for semantic and relevance matching. So extracting STR features is no longer required to achieve the best performance in table retrieval. This is especially important since features like bag-of-entities and bag-of-categories [293] are not always available.

7.4 Summary

We have shown that a hybrid deep model that combines a semantic similarity component and a query relevance component outperforms the best previously published results in table retrieval (STR) [293], achieving up to 9.96% improvement in NDCG@5 score. Consistent with what has been shown in document retrieval, we show that DRSMM, which is a supervised-based method, leads to better ranking results than unsupervised-based methods. Furthermore, we have demonstrated how our approach can be used on tables for which fewer metadata features are available than those required by STR. We have shown that adding the LTR features to our system helps less than adding information about the data values in the table using row and column summaries. This suggests that the specialized LTR [18, 10] and STR [293] features are not useful once one has developed a high quality model for table retrieval.

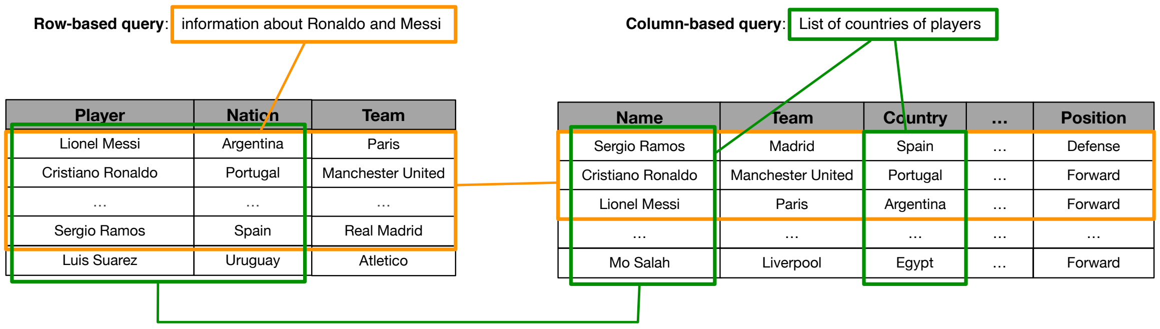
The structural information in DSRMM is captured using summary vectors both in terms of rows and columns. There are two major drawbacks for DSRMM. First, the row and column vectors are computed independently of the context of the data table and query using traditional pretrained word embeddings. A context-independent representation for rows and columns in DSRMM is unable to capture the relationship between the structured form of a data table, and the textual form of both metadata and queries. Second, the row and column vectors are computed using mean pooling, so that the data values are treated equally in summary vectors. Depending on the context of the data table defined by both the metadata and the user's query, each data value should have different contributions in both rows and columns. In the next chapter, we propose a new model that captures the structural information better than DSRMM by learning context-aware representations for rows and columns where we capture different contributions of data values to the final row- and column-level embeddings.

Chapter 8

Structure-aware BERT for Table Search and Matching

8.1 Introduction

Users can search for datasets using a keyword-based query as in document retrieval. This scenario can be seen as ad-hoc table retrieval where multiple methods have been proposed in the literature. Additionally, users can look for similar data tables on the web to an existing table corpus. This can be seen as a query by example scenario or content-based table retrieval, which is similar to content-based image retrieval (CBIR) [59, 250, 300], where the query and the object queried are both data tables. Another table-related task, that requires a table matching phase, is table similarity [85] in which the objective is to predict a binary semantic similarity between two tables. A table similarity algorithm can be used as a core component in multiple tasks such as table classification and clustering [138, 280], table fusion [78] and finding related tables [53]. We consider content-based table retrieval and table similarity as two instances of table matching. The research question (**RQ1**) consists of investigating a single representation for a data table that can be used in both table search and table matching. Figure 8.1 depicts both row/column-based matching between tables and row/column-based queries. In the former case, Figure 8.1 shows how columns and rows are matched to capture the semantic similarity between tables. In the latter case, Figure 8.1 shows two examples of keyword-based table retrieval where the query is a simple and unstructured natural language sequence. The row-based


Row-based query: information about Ronaldo and Messi
Column-based query: List of countries of players

Player	Nation	Team
Lionel Messi	Argentina	Paris
Cristiano Ronaldo	Portugal	Manchester United
...	...	...
Sergio Ramos	Spain	Real Madrid
Luis Suarez	Uruguay	Atletico

Name	Team	Country	...	Position
Sergio Ramos	Madrid	Spain	...	Defense
Cristiano Ronaldo	Manchester United	Portugal	...	Forward
Lionel Messi	Paris	Argentina	...	Forward
...	...	...	...	...
Mo Salah	Liverpool	Egypt	...	Forward

Figure 8.1: Multiple scenarios for table matching and keyword-based table retrieval. Row- and/or column-based matching captures the semantic similarity between tables. Keyword-based queries can match table rows as in the row-based query example, and/or columns as in the column-based query example.

query is relevant to multiple rows of a table, and the column-based query contains keywords related to a subset of attributes from a table. A data table is composed of textual and structural information. First, previous methods decouple the textual information from the structural information of a data table by training a separate model for each field, as in TabSim [85]. Second, the table content is summarized using context-independent row and column summary vectors that are computed using the mean pooling operation, as in DSRMM [237]. Therefore, to answer (**RQ1**), we should study how to fuse the textual and structural information of a data table to produce context-aware representations for both textual and tabular content of a data table (**RQ1.1**). We should also study how to capture the contribution of each data value to the representations of a given row and column (**RQ1.2**). After extracting the structural and textual features of a data table, the research question (**RQ2**) consists of how to incorporate these features into deep learning-based models to solve table-related downstream tasks.

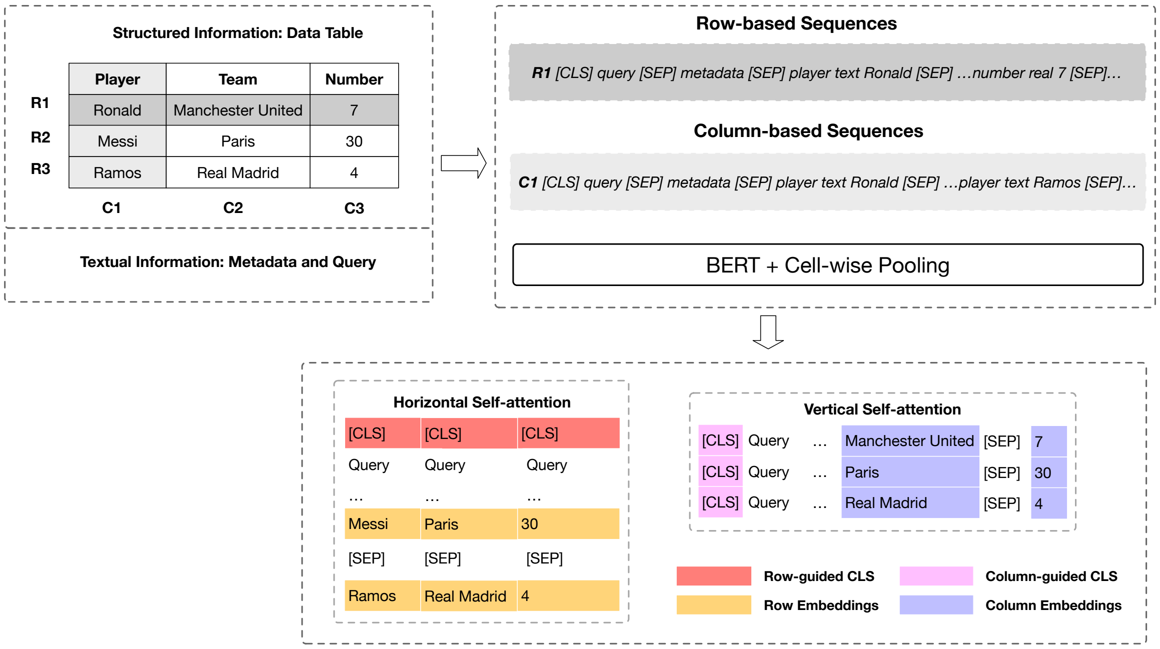
In this chapter, we propose a new model, called ***Structure-aware BERT*** (StruBERT), that fuses the textual and structural information of a data table to produce a context-aware representation for both textual and tabular content of a table. In general, a table can be viewed as a row- and column-based structure and rows and columns should contribute to both (1) the relevance score in table matching where rows and columns of a table pair are matched, and (2) to the retrieval score in keyword-based table retrieval where table content is considered as a relevant field to the keywords of a query. Based on the observations from matching cases in Figure

8.1, we propose a unified model that produces both row- and column-based features to predict the semantic matching between structured/unstructured query and structured table. Inspired by TaBERT [277], our proposed model produces four feature vectors that correspond to the joint representations of the structural and textual information of a table. Two fine-grained features represent the context-aware embeddings of rows and columns, where both horizontal and vertical attentions are applied over the column- and row-based sequences, respectively. Two coarse-grained features capture the textual information from both row- and column-based views of a data table. These features are incorporated into a new end-to-end ranking model, called miniBERT, that is formed of one layer of Transformer [246] blocks, and operates directly on the embedding-level sequences formed from StruBERT features to capture the cross-matching signals of rows and columns.

In summary, we make the following contributions: (1) We propose a new structure-aware BERT model, called StruBERT, that fuses the structural and textual information of a data table to produce four context-aware features: two fine-grained structure- and context-aware representations for rows and columns, and two coarse-grained representations for row- and column-guided [CLS] embedding. (2) We propose a new ranking model, called miniBERT, that operates directly on the embedding-level sequences formed from StruBERT features to solve three table-related downstream tasks: keyword- and content-based table retrieval, and table similarity. (3) We evaluate over three datasets, and demonstrate that our new method outperforms the state-of-the-art baselines, and generalizes to multiple table-related downstream tasks.

8.2 Table views

The primary input to our model is a table T_j that has s rows and l columns. Each table has two forms of information. The first form is the structural information which is composed of headers and data values. A table can be seen as a 2D matrix of cells, and for the purpose of explanation, we assume that the first row corresponds to the headers $c_1, c_2, \ldots, c_l$, and the remaining $s-1$ rows are data values. The i-th column of T_j has the values $v_{1i}, v_{2i}, \ldots, v_{(s-1)i}$. The second form of information is the textual information which corresponds to the context fields of a table. Several text fields can be used to describe the table such as the caption, the title of the page and section that contain the table, etc. We denote these context fields by the metadata which



Structured Information: Data Table
Player
Team
Number
R1
Ronald
Manchester United
7
R2
Messi
Paris
30
R3
Ramos
Real Madrid
4
C1
C2
C3
Textual Information: Metadata and Query
Row-based Sequences
R1 [CLS] query [SEP] metadata [SEP] player text Ronald [SEP] …number real 7 [SEP]…
Column-based Sequences
C1 [CLS] query [SEP] metadata [SEP] player text Ronald [SEP] …player text Ramos [SEP]…
BERT + Cell-wise Pooling
Horizontal Self-attention
Vertical Self-attention
Row-guided CLS
Column-guided CLS
Row Embeddings
Column Embeddings

Figure 8.2: Column- and row-based sequences are formed from the structural and textual information of the table. The sequences are encoded using BERT+cell-wise pooling. Horizontal and vertical self-attentions are applied to the encoded column and row sequences, respectively to obtain four feature vectors: two fine-grained features (row and column embeddings), and two coarse-grained features (row- and column-guided [CLS] embeddings).

forms the textual information of a table. In the case of keyword-based table retrieval, the query is considered as an additional form of textual information because the final representations of StruBERT should capture early interactions between the table and query as in interaction-based retrieval models [95, 80, 179] that have achieved better results than the representation-based models [170]. By learning early interactions between the table and keyword-based query, StruBERT produces structure- and context-aware features, where the query is part of the context.

As shown in Figure 8.2, we propose forming two sets of sequences, denoted by column- and row-based sequences, that are formed on the basis of column- and row-based views, respectively, of a given data table. Yin et al. [277] proposed a row linearization to form sequences from a data table in order to solve the semantic parsing over tables task. Inspired by that, we incorporate row linearization to form row-based sequences, and we propose a column linearization to form column-based sequences.

Given that T_j has l columns, we form l column-based sequences. The i-th column-based sequence is given by:

$$\tilde{c_i} = c_i t_i v_{1i}[SEP]c_i t_i v_{2i}[SEP] \dots [SEP]c_i t_i v_{(s-1)i}[SEP] \tag{8.1}$$

where $t_i \in [real, text]$ is the type of c_i. For example, the first column in the data table shown in Figure 8.2 has a type *text*, and the third column has a type *real*. We use the first column of the table in Figure 8.2 to illustrate an example of a column-based sequence:

$$player\ text\ Ronaldo\ [SEP]\ player\ text\ Messi\ [SEP]\ \dots$$

We denote the set of column-based sequences by $\tilde{\mathcal{C}} = \{\tilde{c_1}, \tilde{c_2}, \dots, \tilde{c_l}\}$. Similarly, we form $s-1$ row-based sequences. The i-th row-based sequence is given by:

$$\tilde{r_i} = c_1 t_1 v_{i1}[SEP]c_2 t_2 v_{i2}[SEP] \dots [SEP]c_l t_l v_{il}[SEP] \tag{8.2}$$

We use the first row of the data table in Figure 8.2 to illustrate an example of a row-based sequence:

$$player\ text\ Ronaldo\ [SEP]\ team\ text\ Juventus\ [SEP]\ \dots$$

We denote the set of row-based sequences by $\tilde{\mathcal{R}} = \{\tilde{r_1}, \tilde{r_2}, \dots, \tilde{r_{(s-1)}}\}$. $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{R}}$ capture only the structural information of T_j. To incorporate the textual information into the structure-based sequences, we concatenate the textual information with each sequence from the structure-based sequences $\tilde{\mathcal{C}} \cup \tilde{\mathcal{R}}$ using the [CLS] and [SEP] tokens of BERT. Given that the textual information Te_j of T_j is formed of p fields $f_1, f_2, \dots, f_p$, the new structure- and context-aware sequences are given by:

$$\overline{c_i} = [CLS]\widetilde{Te_j}[SEP]\tilde{c_i}[SEP] \tag{8.3}$$

$$\overline{r_i} = [CLS]\widetilde{Te_j}[SEP]\tilde{r_i}[SEP] \tag{8.4}$$

where:

$$\widetilde{Te_j} = f_1[SEP]f_2[SEP] \dots [SEP]f_p \tag{8.5}$$

We denote the column- and row-based structure- and context-aware sequences by $\bar{\mathcal{C}} = \{\overline{c_1}, \overline{c_2}, \ldots, \overline{c_l}\}$, and $\overline{\mathcal{R}} = \{\overline{r_1}, \overline{r_2}, \ldots, \overline{r_{(s-1)}}\}$, respectively.

8.3 StruBERT model

Figure 8.2 presents the architecture of StruBERT which is composed of two phases: sequence encoding and self-attention over encoded sequences.

8.3.1 Sequence encoding

To capture the dependencies between the textual information and data values in each sequence from $\bar{\mathcal{C}} \cup \overline{\mathcal{R}}$, BERT is used as a sequence encoder which produces contextualized embeddings for each token in the tokenized sequence using BERT tokenizer. BERT is preferred over a recurrent architecture because BERT is composed of Transformer blocks that capture long-range dependencies with self-attention better than recurrent architectures [246], and is pretrained on large textual data.

After row and column linearization and BERT tokenization, each cell has multiple tokens. To compute a single embedding for each cell, we incorporate cell-wise average pooling [277] after the BERT encoding step to pool over the contextualised tokens for each cell defined by [header_name type cell_content]. BERT is composed of L layers of Transformer blocks. The cell-wise average pooling is applied on the contextualized embedding that is obtained from the last layer. The contextualized embedding of the column-based sequence $\overline{c_i}$ is given by:

$$\boldsymbol{\overline{c_i}} = [\boldsymbol{CLS}]\widetilde{\boldsymbol{Te_j}}[\boldsymbol{SEP}]\boldsymbol{v_{1i}}[\boldsymbol{SEP}]\ldots[\boldsymbol{SEP}]\boldsymbol{v_{(s-1)i}}[\boldsymbol{SEP}] \tag{8.6}$$

where:

$$\boldsymbol{v_{ki}} = \frac{\sum_{w \in BertTok(c_i t_i v_{ki})} \boldsymbol{h_w^L}}{|BertTok(c_i t_i v_{ki})|}; \;\; k = 1, 2, \ldots, s-1 \tag{8.7}$$

$BertTok(c_i t_i v_{ki})$ represents the tokens that are obtained after tokenizing the sequence $c_i t_i v_{ki}$ using BERT tokenizer, and $\boldsymbol{h_w^L} \in \mathbb{R}^d$ is the contextualized embedding of dimension d from the L-th layer of BERT for the token $w \in BertTok(c_i t_i v_{ki})$. Similarly, the cell-wise average pooling is used to compute the contextualized embedding for the row-based sequence $\overline{r_i}$, denoted by $\boldsymbol{\overline{r_i}}$. We denote the column- and row-based contextualized embeddings that are obtained after BERT and cell-wise average pooling by $\boldsymbol{\bar{\mathcal{C}}} = \{\boldsymbol{\overline{c_1}}, \boldsymbol{\overline{c_2}}, \ldots, \boldsymbol{\overline{c_l}}\}$ and $\boldsymbol{\overline{\mathcal{R}}} = \{\boldsymbol{\overline{r_1}}, \boldsymbol{\overline{r_2}}, \ldots, \boldsymbol{\overline{r_{(s-1)}}}\}$, respectively.

8.3.2 Horizontal and Vertical self-attention

Self-attention is incorporated into StruBERT for two reasons. First, the contextualized embeddings in $\bar{\boldsymbol{\mathcal{C}}}$ capture independent column-level structural and textual information, and ignore the row-level dependency as a result of tabular structure. The same conclusion applies for $\overline{\mathcal{R}}$ where column-level dependency is not captured in the row-level embeddings. Second, cell values are not equally important for the representations of rows and columns. We incorporate vertical self-attention [277] to operate over row-based embeddings to produce column embeddings, and we propose a horizontal self-attention that operates over column-based embeddings to form row embeddings. Both attentions are similar to the Transformer [246], and the naming of horizontal and vertical attention comes from the orientation of input sequences to attention blocks.

Horizontal self-attention

To capture the row-level dependency between the column-based contextualized embeddings of $\bar{\boldsymbol{\mathcal{C}}}$, we propose a multi-head horizontal self-attention that operates on horizontally aligned tokens from the column-based embeddings as shown in Figure 8.2. The horizontal self-attention is formed of H layers of Transformers, and we use the output of the last layer as the row-level self-attention representation. We produce two types of features from the horizontal self-attention step after applying row-level average pooling. First, we obtain $s - 1$ row embeddings which can be seen as fine-grained structure- and context-aware features. Second, by averaging the [CLS] embedding from each column, we produce a row-guided [CLS] which represents a coarse-grained structure and context-aware feature. In conclusion, the horizontal self-attention features are based on interpreting the data table as a column-based structure, followed by row-level dependency.

Vertical self-attention

Similarly, a data table can be interpreted as a row-based structure, followed by column-level dependency. In this case, V layers of vertical self-attention [277] operate on the row-based contextualized embeddings of $\overline{\boldsymbol{\mathcal{R}}}$. We also obtain two types of features from the vertical self-attention. First, we obtain l fine-grained column embeddings by averaging the last output of the vertical self-attention over the vertically aligned tokens from the row-based embeddings. Second, we obtain a coarse-grained column-guided [CLS] embedding that interprets the data table as a row-based structure, followed by column-level dependency.

In conclusion, StruBERT generates four structure- and context-aware features: two fine-grained features which are the contextualized row and column embeddings, denoted by $\boldsymbol{E_r} \in \mathbb{R}^{(s-1)\times d}$ and $\boldsymbol{E_c} \in \mathbb{R}^{l\times d}$, respectively, and two coarse-grained features which are the row- and column-guided [CLS] embedding, denoted by $[\boldsymbol{CLS}]_{\boldsymbol{r}} \in \mathbb{R}^d$ and $[\boldsymbol{CLS}]_{\boldsymbol{c}} \in \mathbb{R}^d$, respectively.

8.4 StruBERT features in downstream tasks

We integrate StruBERT as a feature extractor F into end-to-end architectures to solve table-related downstream tasks. In this section, we address the tasks of table search and table matching, and we show how to map StruBERT features to classification or retrieval score depending on the task.

8.4.1 Table matching

In table matching tasks, both the query and the queried object are data tables. The neural ranking model should capture the semantic similarity between the structural and textual information of table pairs in order to predict the relevance score. To this end, we propose a Siamese [15]-based model that predicts the relevance score of a table pair (T_i, T_j). In table matching, the textual information of each table contains only the metadata because the keyword-based query is absent. Structure- and context-aware features are extracted from each table using StruBERT:

$$F(T_i, T_j) = (StruBERT(T_i), StruBERT(T_j))$$

$$F(T_i, T_j) = ((\boldsymbol{E_r^i}, \boldsymbol{E_c^i}, [\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{i}}, [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{i}}), (\boldsymbol{E_r^j}, \boldsymbol{E_c^j}, [\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{j}}, [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{j}}))$$

After extracting features from each table using StruBERT, we obtain coarse- and fine-grained features for each table. We propose a ranking model that captures the semantic similarities within the fine-grained features $((\boldsymbol{E_r^i}, \boldsymbol{E_c^i})$ and $(\boldsymbol{E_r^j}, \boldsymbol{E_c^j}))$, and coarse-grained features $(([\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{i}}, [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{i}})$ and $([\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{j}}, [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{j}}))$.

Cross-matching of fine-grained features:

To capture cross-matching signals of row and column embeddings for table pairs, we propose a model, called miniBERT, that operates directly on the embedding-level sequences of fine-grained features of StruBERT. miniBERT is composed of three trainable vectors $[\boldsymbol{REP}]_{\boldsymbol{c}} \in \mathbb{R}^d$, $[\boldsymbol{REP}]_{\boldsymbol{r}} \in \mathbb{R}^d$, and $[\boldsymbol{SEP}] \in \mathbb{R}^d$, and 1 layer of Transformer blocks with 4 attention heads.[1] The input to miniBERT for column-based cross-matching of a table pair (T_i, T_j) is shown in Figure 8.3. $[\boldsymbol{REP}]_{\boldsymbol{c}}$ is introduced to aggregate the matching signals between $\boldsymbol{E^i_c}$ and $\boldsymbol{E^j_c}$. We form the embedding-level sequence for the column embeddings of a table pair (T_i, T_j):

$$M_{c_ic_j} = [\boldsymbol{REP}]_{\boldsymbol{c}} \oplus \boldsymbol{E^i_c} \oplus [\boldsymbol{SEP}] \oplus \boldsymbol{E^j_c} \tag{8.8}$$

where $[\boldsymbol{SEP}]$ is used to separate $\boldsymbol{E^i_c}$ and $\boldsymbol{E^j_c}$. As in BERT, we sum three different embeddings to obtain the input embeddings to miniBERT. As shown in Figure 8.3, in addition to the column embeddings, the segment embeddings are used to indicate the column embeddings that belong to T_i and T_j, and the position embeddings are used to encode the position of each vector in $M_{c_ic_j}$. The position embedding of $[\boldsymbol{REP}]_{\boldsymbol{c}}$ is in particular useful to indicate that the final hidden state from the first position aggregates the embedding-level sequence $M_{c_ic_j}$. So, miniBERT takes the embedding-level sequence, that is formed by summing the column, segment and position embeddings, as input, then miniBERT outputs the hidden state of $[\boldsymbol{REP}]_{\boldsymbol{c}}$ from the Transformer block, denoted by $miniBERT([\boldsymbol{REP}]_{\boldsymbol{c}})$, that captures the bidirectional cross-attention between $\boldsymbol{E^i_c}$ and $\boldsymbol{E^j_c}$.

Similarly, we use miniBERT to compute the hidden state of $[\boldsymbol{REP}]_{\boldsymbol{r}}$, denoted by $miniBERT([\boldsymbol{REP}]_{\boldsymbol{r}})$, from the embedding-level sequence input for rows defined by:

$$M_{r_ir_j} = [\boldsymbol{REP}]_{\boldsymbol{r}} \oplus \boldsymbol{E^i_r} \oplus [\boldsymbol{SEP}] \oplus \boldsymbol{E^j_r} \tag{8.9}$$

There are mainly two advantages from using miniBERT as a ranking model on top of the StruBERT features. First, a row- or column-based permutation for a table does not change the meaning of the table. The self-attention of the Transformer blocks in miniBERT is particularly useful where each embedding attends to all embeddings

[1]We tried to increase the number of layers and attention heads, but we did not notice an improvement in the reported evaluation metrics.

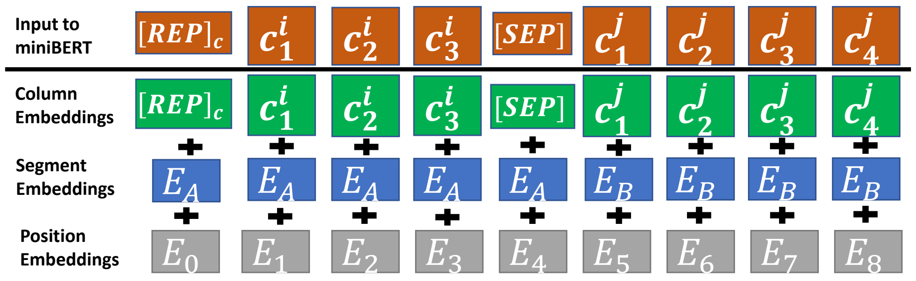


Figure 8.3: Embedding-level sequence input of miniBERT for cross-matching of columns. The input to miniBERT is the sum of three embeddings: column, segment, and position embeddings. In this example, $\boldsymbol{E_c^i} \in \mathbb{R}^{3\times d}$ is composed of $c_k^i \in \mathbb{R}^d, k \in [1, 2, 3]$, and $\boldsymbol{E_c^j} \in \mathbb{R}^{4\times d}$ is composed of $c_k^j \in \mathbb{R}^d, k \in [1, 2, 3, 4]$.

in the column- and row-based embedding-level sequences regardless of the position information. Second, evaluating the semantic similarity between tables is not based only on one-to-one mapping between columns or rows. For example, one column from T_i can summarize the information that is present in three columns from T_j. The attention weights in the attention heads of miniBERT are valuable to capture many-to-many relationships between columns (rows) of a table pair by aggregating information both within and across table columns (rows).

Cross-matching of coarse-grained features:

Similarly to the fine-grained features, we construct the cross-matching features between the coarse-grained features of T_i and T_j. We define the interaction vectors $\mathcal{F} = \{\boldsymbol{F_{r_i r_j}}, \boldsymbol{F_{c_i c_j}}\}$, where $\boldsymbol{F_{r_i r_j}}$, and $\boldsymbol{F_{c_i c_j}}$ denote the interactions of $[\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{i}}$-$[\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{j}}$, and $[\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{i}}$-$[\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{j}}$, respectively, and the elements of each vector are computed using pointwise multiplication between the embeddings of the corresponding row- and column-guided [CLS]:

$$\boldsymbol{F_{r_i r_j}} = [\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{i}} \odot [\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{j}} \; ; \; \boldsymbol{F_{c_i c_j}} = [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{i}} \odot [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{j}} \tag{8.10}$$

Ranking layer:

The fine- and coarse-grained features are used as input to a ranking layer to predict the relevance score of a table pair. The final feature vector of a table pair (T_i, T_j) is given by:

$$\Phi(T_i, T_j) = \boldsymbol{F_{r_i r_j}} \oplus \boldsymbol{F_{c_i c_j}} \oplus miniBERT([\boldsymbol{REP}]_{\boldsymbol{r}}) \oplus miniBERT([\boldsymbol{REP}]_{\boldsymbol{c}}) \tag{8.11}$$

A final linear layer is used to predict the relevance score of the table pair (T_i, T_j) using $\Phi(T_i, T_j)$:

$$g(T_i, T_j) = \omega_r^T \Phi(T_i, T_j) + b_r \tag{8.12}$$

where ω_r and b_r are the parameters of the linear layer.

8.4.2 Keyword-based table retrieval

The query q is composed of several keywords, $q_1, q_2, \ldots, q_m$ where m is the length of the query and q_l is the l-th token of q, and the queried object is a data table T_i from a table corpus $\mathcal{C}$. In addition to the table's metadata, the textual information Te_i contains the query q so that the outputs of StruBERT capture early interactions between the query and the structural and textual information of a data table. We use the same notations of the table matching case, and we denote the outputs of StruBERT for a given query-table pair (q, T_i) by: $\boldsymbol{E_r^i}, \boldsymbol{E_c^i}, [\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{i}}, [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{i}}$. We apply miniBERT to the single embedding-level sequences defined by:

$$\begin{aligned} M_{r_i q} &= [\boldsymbol{REP}]_{\boldsymbol{r}} \oplus \boldsymbol{E_r^i(q)} \oplus [\boldsymbol{SEP}] \\ M_{c_i q} &= [\boldsymbol{REP}]_{\boldsymbol{c}} \oplus \boldsymbol{E_c^i(q)} \oplus [\boldsymbol{SEP}] \end{aligned} \tag{8.13}$$

where $\boldsymbol{E_r^i}$ and $\boldsymbol{E_c^i}$ are function of q because $q \in T_{e_i}$ in the case of keyword-based table retrieval. We use the final hidden states of $[\boldsymbol{REP}]_{\boldsymbol{r}}$ and $[\boldsymbol{REP}]_{\boldsymbol{c}}$ that are obtained from miniBERT as the row- and column-based aggregate for the query-table pair (q, T_i), respectively. A query-table pair (q, T_i) is represented using four feature vectors: row and column outputs from miniBERT and row- and column-guided [CLS] embeddings $([\boldsymbol{CLS}]_{\boldsymbol{r}}^{\boldsymbol{i}}, [\boldsymbol{CLS}]_{\boldsymbol{c}}^{\boldsymbol{i}})$. We concatenate these features to obtain the final representation for (q, T_i), which is used as input to a linear layer to predict the relevance score of the query-table pair (q, T_i).

8.5 Evaluation

8.5.1 Baselines

Keyword-based table retrieval

For evaluation on the keyword-based table retrieval, we compare against the following baselines:

MultiField-BM25: In a multi-field ranking scenario, a table is defined using multiple fields. MultiField-BM25 combines BM25 scores for multi-field tables.

MCON [241]: This baseline is based on new word embeddings for attribute tokens of tables. When calculating the retrieval score of MCON, we use the MaxTable ranking method which was shown to be the best ranking method for unsupervised table retrieval [241]. MaxTable is a late fusion similarity model that finds the closest table token to each query token using cosine similarity, and then sums over these similarities.

STR [293]: Multiple embedding-based features are computed for a table and query, then different matching strategies are used to generate the ranking features from the embeddings. A random forest is used to predict the relevance score of a query-table pair.

BERT-Row-Max [36]: The [CLS] embedding of the sequence formed from the query and table is used to predict the relevance score of a query-table pair.

DSRMM [237]: It is a joint model that captures both semantic and relevance matching signals from a query-table pair to predict a real-valued relevance score.

MutiEm-RGCN [238]: It is a two-phased graph-based model that is used for table retrieval. Multiple embeddings are learned from a knowledge graph, then used to represent the table and query for keyword-based table search.

TaBERT [277]: A model that is originally proposed for semantic parsing over tables. We use the embedding of the [CLS] token from the last layer of the vertical self-attention as input to a MLP layer.

Table matching

For evaluation in table matching, we compare against the following baselines:

embedding+MLP: A table is flattened and concatenated with the metadata to form a single document for each table. Then, the mean of word embeddings using Glove is calculated for each table. The final ranking feature is computed using pointwise multiplication between the embeddings of tables, then forwarded to a MLP layer to predict the relevance score.

TF-IDF+MLP: TableRank [147] computes Term Frequency-Inverse Document Frequency (TF-IDF) for tables. The TF-IDF score is computed using the metadata and values of a given table, instead of the document that contains the table. A MLP layer is used instead of the cosine similarity to predict the semantic matching score.

TabSim [85]: Two separate neural network models are introduced to form the representations of a table: one model extracts the embedding from the caption, and a second model extracts column embeddings from the data values.

TaBERT [277]: A TaBERT-based Siamese model is used to evaluate the semantic similarity between tables.

StruBERT (KP): This baseline is a variation of our method that uses a kernel pooling (KP) [263]-based ranking model on top of StruBERT features. Kernel pooling is the main component of strong ranking models [263, 51], and we adapt kernel pooling for cross-matching of fine-grained features. We construct the interaction matrices $I = \{I_{r_i r_j}, I_{c_i c_j}\}$, where $I_{r_i r_j}$, and $I_{c_i c_j}$ denote the interactions of $\boldsymbol{E_r^i}$-$\boldsymbol{E_r^j}$ and $\boldsymbol{E_c^i}$-$\boldsymbol{E_c^j}$ respectively, and the elements of each matrix are computed using cosine similarity between the embeddings of the corresponding rows and columns. To summarize each interaction matrix into a fixed-length feature vector, we use kernel pooling to extract soft-match signals between different fields of T_i and T_j. A linear layer is used to map the KP-based feature vector to a relevance score.

StruBERT (CNN): This baseline is a variation of our method that uses Convolutional Neural Networks (CNN) on top of StruBERT features. This baseline is based on the interaction tensor, denoted by $\mathcal{S}$, which is computed using pointwise multiplication between pairwise column (row) embeddings of a table pair. Inspired by DeepRank [179], we use one layer CNN filters with all possible combinations of widths and heights that are applied to $\mathcal{S}$:

$$h_{i,j}^{(\kappa)} = \sum_{s=1}^{\gamma} \sum_{t=1}^{\gamma} \left(\sum_{l=1}^{d} w_{s,t}^{l(\kappa)} \cdot \mathcal{S}_{i:i+s,j:j+t}^{(l)} \right) + b_{s,t}^{(\kappa)},\ \kappa = 1, \cdots, K \tag{8.14}$$

where γ is the maximum size of a CNN filter, $\mathcal{S}_{i:i+s,j:j+t}^{(l)}$ is a $s \times t$ matrix from the l-th channel of $\mathcal{S}$ starting from i-th row and j-th column, K is the total number of

CNN filters, and $w_{s,t}^{l(\kappa)}$ and $b_{s,t}^{(\kappa)}$ are parameters of CNN. Then, we keep only the most significant matching signal from each feature map to form a single vector.

$$h^{(\kappa)} = \max_{i,j} h_{i,j}^{(\kappa)}, \kappa = 1, \cdots, K \tag{8.15}$$

8.5.2 Experimental Setup

Our model is implemented using PyTorch, with two NVIDIA GeForce GTX 1080. For keyword- and content-based table retrieval, the parameters of our model are updated using a mean square error pointwise loss between predicted and groundtruth relevance scores, and for table similarity, we use the cross-entropy loss function. We use the Adam [120] optimizer for gradient descent to minimize the loss functions. The dimension d is equal to 768. The number of Transformer layers H and V in the horizontal and vertical self-attention, respectively, are equal to 3. We set the maximum length of each sequence to 256. In StruBERT, the BERT-base-uncased and the vertical self-attention are initialized using TaBERT$_{Base}(K = 3)$[2] which is pretrained using content snapshots with 3 rows. Such pretraining requires high-memory GPUs that are not currently possessed by our team; therefore, we randomly initialize the horizontal self-attention [3] so that the row-based dependencies are only captured during fine-tuning on the target dataset. We expect an increase in the results with pretraining the horizontal self-attention on a similar task to the Masked Column Prediction (MCP) from TaBERT [277] (in our case, the pretraining task should be a Masked Row Prediction). We leave pretraining the horizontal self-attention as a future direction.

8.5.3 Experimental results

We report results using five-fold cross validation. For keyword-based table retrieval, we use the same splits as Chen et al. [36] to report results on the five-fold cross validation for our method and baselines. We evaluate the performance of our proposed method and baselines on the keyword- and content-based table retrieval tasks using Normalized Discounted Cumulative Gain (NDCG) [107], Mean Reciprocal Rank

[2]https://github.com/facebookresearch/TaBERT

[3]We tried to initialize the horizontal self-attention using the vertical self-attention from TaBERT [277], but we did not notice an improvement in the reported metrics.

(MRR), and Mean Average Precision (MAP). All evaluation metrics results are reported using the TREC evaluation software, trec_eval[4]. We evaluate the performance of our method and baselines on the table similarity task using macro-averaged precision (P), recall (R) and F-score, and accuracy of predictions on the testing set. To test significance, we use the paired Student's t-test and write † to denote significance at the 0.05 level over all other methods.

Table similarity results

Table 8.1(a) shows the performance of different approaches on the PMC collection. We show that our proposed method StruBERT outperforms the baselines for all evaluation metrics. By incorporating the structural and textual features into a cross-matching based ranking model, we were able to capture the semantic similarity between tables both in term of tabular content and metadata, and this leads to an increase in evaluation metrics compared to baselines that either ignore the structural information or treat the textual and structural information separately. Considering the table as a single document in TF-IDF and embedding baselines lead to the lowest results which indicates that the structural similarity between tables is an important factor in table similarity. The results on this dataset show a clear advantage from using embedding-based features (traditional or contextualized) compared to term frequency features that are based on exact matching. StruBERT (fine) and StruBERT (coarse) show ablation study results for predicting semantic similarity using only fine- and coarse-grained features, respectively. By combining both categories of features, we achieve higher evaluation metric results.

Table 8.1(b) shows the performance of the different approaches on the WikiTables. Consistent with PMC, our results on the WikiTables show the importance of the structure- and context-aware features in improving the table similarity prediction. Table similarity results on WikiTables and PMC show that StruBERT achieves significant improvements in the evaluation metrics of two data table collections from different domains, which supports the generalization characteristic of our proposed method.

[4]https://trec.nist.gov/trec_eval/trec_eval.8.1.tar.gz

Method Name	Macro-P	Macro-R	Macro-F	Accur.
tfidf+MLP	0.7834	0.6735	0.6529	0.6951
embedding+MLP	0.8496	0.7710	0.7736	0.7931
tfidf+embedding+MLP	0.8736	0.8381	0.8447	0.8506
TabSim [85]	0.8865	0.8545	0.8613	0.8705
TaBERT [277]	0.9109	0.9024	0.9055	0.9067
StruBERT (fine)	0.9208	0.9058	0.9104	0.9124
StruBERT (coarse)	0.9276	0.9154	0.9194	0.9210
StruBERT (KP)	0.9148	0.9060	0.9091	0.9109
StruBERT (CNN)	0.9293	0.9164	0.9205	0.9224
StruBERT	**0.9321**†	**0.9284**†	**0.9300**†	**0.9310**†

(a) PMC

Method Name	Macro-P	Macro-R	Macro-F	Accur.
tfidf+MLP	0.6256	0.5022	0.3559	0.5378
embedding+MLP	0.8429	0.8419	0.8423	0.8433
tfidf+embedding+MLP	0.8632	0.8554	0.8574	0.8594
TabSim [85]	0.8480	0.8458	0.8466	0.8478
TaBERT [277]	0.9696	0.9626	0.9649	0.9653
StruBERT (fine)	0.9850	0.9852	0.9851	0.9852
StruBERT (coarse)	0.9838	0.9816	0.9825	0.9826
StruBERT (KP)	0.9733	0.9713	0.9722	0.9724
StruBERT (CNN)	0.9782	0.9737	0.9753	0.9756
StruBERT	**0.9945**†	**0.9938**†	**0.9941**†	**0.9942**†

(b) WikiTables

Table 8.1: Table similarity results using a classification threshold equal to 0.5.

Content-based table retrieval results

Table 8.2 shows the content-based table retrieval results. The StruBERT model that combines fine- and coarse-grained features achieves a 7.96% improvement in terms of NDCG@5 upon TaBERT that only uses the [CLS] embedding obtained from the vertical self-attention. We also report ablation study results where we predict the semantic similarity between tables using only fine- or coarse-grained features. Both categories of features lead to better retrieval results than the baselines, and by combining both the fine- and coarse-grained features, we capture the textual and structural similarities between tables. An important step in table similarity assessment is the order-invariant cross-matching between columns (rows) of tables which is satisfied

Table 8.2: Content-based table retrieval results on the query by example dataset [291].

Model	**NDCG@5**	**MRR**	**MAP**
BM25	0.5369	0.5832	0.5417
DSRMM [237]	0.5768	0.6193	0.5914
TabSim [85]	0.5739	0.6056	0.5932
TaBERT [277]	0.5877	0.6120	0.5942
StruBERT (fine)	0.6015	0.6419	0.6091
StruBERT (coarse)	0.6140	0.6478	0.6142
StruBERT (KP)	0.5990	0.6200	0.5959
StruBERT (CNN)	0.6177	0.6378	0.6179
StruBERT	**0.6345**†	**0.6601**†	**0.6297**

using miniBERT as a ranking model on top of StruBERT features.

Our approach uses a novel miniBERT model to map StruBERT features to a relevance score. We investigate the performance of alternative ranking models when given StruBERT features. Table 8.2 shows the results of comparing miniBERT against StruBERT (KP) and StruBERT (CNN) in the case of content-based table retrieval. miniBERT outperforms both baselines in all evaluation metrics. Kernel pooling summarizes the one-to-one matching signals computed in the interaction matrix to a single feature vector. So, StruBERT (KP) does not solve the case of many-to-many matching of rows or columns. On the other hand, by applying CNN to the interaction tensor, we capture the semantic similarity between multiple columns (rows) from a table pair, so StruBERT (CNN) captures the many-to-many matching signals. However, the convolution operation captures local interactions, and is not permutation invariant in the sense that rows and columns could be arbitrarily ordered without changing the meaning of the table. miniBERT deals with both many-to-many matching and the permutation invariant property by taking advantage of self-attention heads.

Keyword-based table retrieval results

Table 8.3 shows the performance of different approaches on the WikiTables collection. We show that our proposed method StruBERT outperforms the baselines for all evaluation metrics. Consistent with what has been shown in ad hoc document retrieval, supervised approaches perform better on ad hoc table retrieval than unsupervised

Table 8.3: Keyword-based table retrieval results on the WikiTables dataset [9].

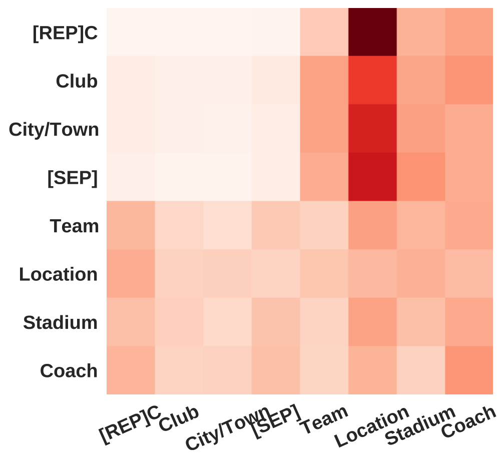
Model	**NDCG@5**	**MRR**	**MAP**
MultiField-BM25	0.4365	0.4882	0.4596
MCON [241]	0.5152	0.5321	0.5193
STR [293]	0.5762	0.6062	0.5711
DSRMM [237]	0.5978	0.6390	0.5992
MultiEm-RGCN [238]	0.5855	0.6253	0.5920
TaBERT [277]	0.6055	0.6462	0.6123
BERT-Row-Max [36]	0.6167	0.6436	0.6146
StruBERT (fine)	0.6000	0.6406	0.6020
StruBERT (coarse)	0.6217	0.6562	0.6225
StruBERT	**0.6393**†	**0.6688**†	**0.6378**

approaches (MultiField-BM25 and MCON). By adding the query to the textual information of a given table, we obtain fine- and coarse-grained features that capture early interactions between the query, and the structural and textual information of a table.

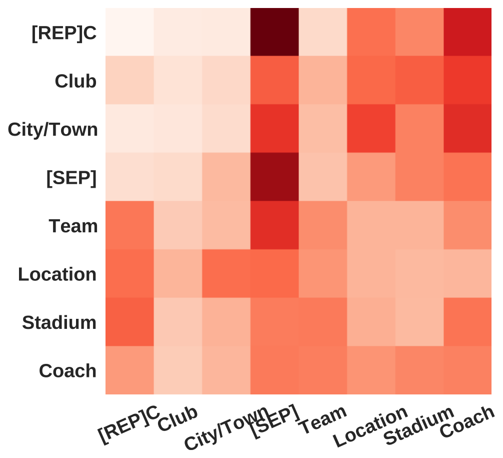
For the ablation study of the keyword-based table retrieval, we notice that summarizing the table and query using the [CLS] token in BERT-Row-Max, TaBERT, and StruBERT (coarse) leads to better results than fine-grained features of StruBERT (fine). This means that there are more natural language queries in the keyword-based table retrieval for WikiTables collection that are relevant to a high level summary of the textual and structural information than the specific details captured by rows and columns. After combining the fine- and coarse-grained features for all query-table pairs, StruBERT captures the semantic similarity between the query and the textual information, and the query and the structural information defined by rows and columns, and this leads to the best retrieval metrics.

miniBERT attention heads

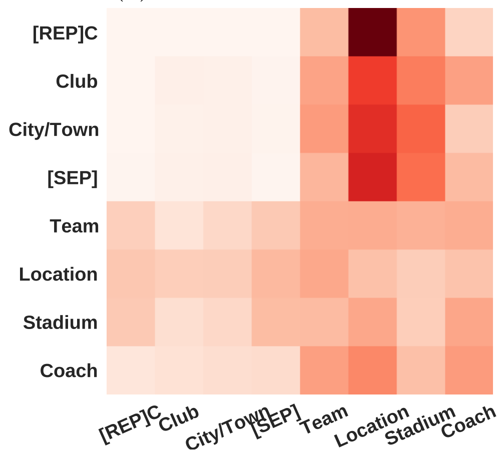
miniBERT is composed of one layer of Transformer blocks with four attention heads. To better understand how miniBERT works, we show the attention heads that correspond to a table similarity case. The first table is composed of the headers *Club* and *City/Town*, and the second table is composed of the headers *Team*, *Location*, *Stadium*, and *Coach*. Figure 8.4 illustrates the four attention heads of miniBERT. Figure 8.4(a) indicates that the 1st attention head focuses on the header *Location*



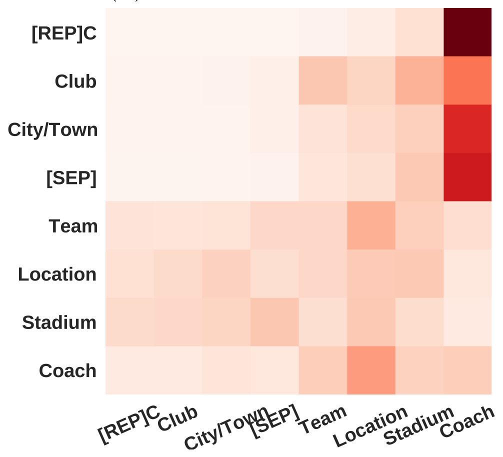
(a) 1st attention head



(b) 2nd attention head



(c) 3rd attention head



(d) 4th attention head

Figure 8.4: Comparison of attention heads between columns of a table pair.

from the second table which attends mainly to the header *City/Town* from the first table and contributes significantly to the embedding $[\boldsymbol{REP}]_{\boldsymbol{c}}$. The second attention head, illustrated in Figure 8.4(b), is more general as it indicates multiple cross matching signals between columns of both tables. The third attention head in Figure 8.4(c) is similar to the 1st attention head with more focus on the header *Stadium* from the second table. This can be explained by the co-occurrence of the header *Stadium* with headers *Club* and *City/Town*. We also observe similar patterns in the 4th attention head that focuses mainly on the header *Coach*. The analysis of the attention heads shows the advantage of using the Transformers blocks to capture the many-to-many relationships between columns of tables.

8.6 Summary

We have shown that a structure-aware BERT-based model, called StruBERT, that fuses the structural and textual information of data tables outperforms the state-of-the-art results in three table-related tasks: keyword- and content-based table retrieval, and table similarity. StruBERT embeddings are integrated into our miniBERT ranking model to predict the relevance score between keyword-based query and table, or between table pairs. Despite being a general model, StruBERT outperforms all baselines on three different tasks, achieving a near perfect accuracy of 0.9942 on table similarity for WikiTables, and improvement in table search for both keyword- and table-based queries with up to 7.96% increase in NDCG@5 score for content-based table retrieval. An ablation study shows that using both fine- and coarse-grained features achieves better results than either set alone. We also demonstrate that using miniBERT as a ranking model for StruBERT features outperforms other common ranking model approaches.

Chapter 9

Conclusions of Dataset Search

9.1 Overview of proposed methods

We proposed multiple unsupervised and supervised methods for dataset search. First, we proposed an unsupervised method, called MCON [241], where we incorporate the contextual information of data tables to learn a new embedding for attribute tokens using an adapted Skip-gram model. Recently, in document retrieval, external knowledge bases and graphs are incorporated into neural ranking models to provide additional embeddings for the query and document. Knowledge graphs contain human knowledge and can be used in neural ranking models to better understand queries and documents. In general, the entity-based representation, that is computed from knowledge bases, is combined with the word-based representation. In addition, the knowledge graph semantics, such as the description and type of entity, provide additional signals that can be incorporated into the neural ranking model to improve retrieval results and generalize to multiple scenarios. Consistent with document retrieval, in dataset search, we learn better embedding for all tokens in data tables by incorporating external semantic and lexical knowledge in MultiEm-RGCN [238].

The unsupervised methods are usually used as an initial ranker to obtain an initial set of relevant datasets where the recall is more important than the precision. A supervised method is then needed to refine the initial set of datasets by improving the precision, and returning only the most relevant dataset to the user. As a first supervised method, we proposed DSRMM [237] which is a deep semantic and relevance matching model. Generally, Semantic and relevance signals are important matching signals in document retrieval. Semantic matching is introduced in multiple text matching tasks, such as natural language inference, and paraphrase identification. Semantic matching, which aims to model the semantic similarity between the query and the document, assumes that the input texts are homogeneous. Semantic matching captures composition and grammar information to match two input texts which are compared in their entirety. In information retrieval, the QA task is a good scenario for semantic matching, where semantic and syntactic features are important to compute the relevance score. On the other hand, semantic matching is not enough for document retrieval, because a typical scenario is to have a query that contains keywords. In such cases, the relevance matching is needed to achieve better retrieval results. Relevance matching is introduced by Guo et al. [80] to solve the case of heterogeneous query and document in ad hoc document retrieval. The query can be expressed by keywords, so a semantic signal is less informative in this case because the composition and grammar of a keyword-based query are not well defined. In addition, the position of a given token in a query has less importance than the strength of the similarity signal. DSRMM combines both the semantic and relevance signals for dataset search. In addition, DSRMM incorporates the structural information of datasets by incorporating summary vectors both in terms of rows and columns.

In DSRMM, the row and column vectors are computed independently of the context of the data table and query using traditional pretrained word embeddings. A context-independent representation for rows and columns in DSRMM is unable to capture the relationship between the structured form of a data table, and the textual form of both metadata and queries. In addition, the row and column vectors are computed using mean pooling, so that the data values are treated equally in summary vectors. Depending on the context of a data table defined by both the metadata and the user's query, each data value should have different contributions in both rows and columns. To overcome these limitations, we proposed a new BERT-based model, called StruBERT [240], that captures the structural information better than DSRMM by learning context-aware representations for rows and columns where we capture different contributions of data values to the final row- and column-level embeddings.

9.2 Effectiveness and efficiency trade-off

9.2.1 Time and memory complexity

When we deploy a dataset search engine, millions of users will input queries to search for relevant datasets from an initial corpus with millions of datasets. Therefore, in the deployment phase, the dataset search engine must be both effective and efficient. StruBERT is effective for dataset search in terms of retrieval metrics. However, StruBERT is based on BERT which has expensive time and memory complexity in document retrieval in general. For example, ranking documents of length m using Transformers, which are the main components of BERT, requires $O(m^2)$ memory and time complexity [122]. In particular, for very long documents, applying self-attention of Transformers is not feasible. So, BERT-based ranking models have a large increase in computational cost and memory complexity over the existing traditional and neural ranking models. A current research direction in document retrieval is the design of efficient and effective deep language model-based ranking architectures. For example, Khattab and Zaharia [116] presented a ranking model that is based on contextualized late interaction over BERT (ColBERT). The proposed model reduces computation time by extracting BERT-based document representations offline, and delays the interaction between query and document representations. RepBERT [287] and RocketQA [195] are other models proposed to solve the passage retrieval task following the same direction of designing a representation-based model with BERT being the main component to map the query and document. As in ColBERT, the objective is to overcome the computation time and memory limitations of the cross-encoding attentions related to the sentence pair setting of BERT. To reduce the processing time of StruBERT, it is possible to delay the interactions between the query and tables by considering the late interaction design. However, there is a risk of losing important early interactions between datasets and queries as in representation-based models. In the same direction of reducing the complexity of BERT-based models, Hofstätter et al. [94] reduced the time and memory complexity of Transformers by considering the local self-attention where a given token can only attend to tokens in the same sliding window. In the particular case of non-overlapping sliding windows of size $w << m$, the time and memory complexity is reduced from $O(m^2)$ to $O(m \times w)$. Recently, Kitaev et al. [122] improved the efficiency of Transformers and proposed the Reformer which is efficient in terms of memory and runs faster for long sequences by reducing the complexity from $O(m^2)$ to $O(m \times log(m))$. The Reformer can be used in StruBERT to achieve the trade-off between high retrieval results and low computation time.

Generally, reducing the memory complexity is still an open research question in document retrieval. First, the document embeddings need to be loaded into the system or GPU memory [110] with a limited size to compute the relevance scores of query-document pairs. Second, the dimension of the embedding is very large compared to the bag-of-word index [286, 264]. Therefore, vector compression methods [108, 73] have been integrated into neural ranking models to compress the embedding index and save computational resources with compressed embeddings of documents. The compression methods include Product Quantization (PQ) [108, 73] and Locality Sensitive Hashing (LSH) [102]. Improving the memory efficiency using index compression can lead to a drop in the performance of the neural ranking model. Zhan et al. [285] proposed a joint optimization of query encoding and PQ in order to maintain effectiveness of neural ranking models while compressing index sizes. The authors showed that an end-to-end training strategy of the encoding and compression steps overcomes the training based on the reconstruction error for many compression methods [108, 73, 82]. PQ can be incorporated in the late interaction design of StruBERT to reduce both the time and memory complexity.

9.2.2 Multi-stage dataset search

A multi-stage ranking architecture for dataset search is suitable for the trade-off between retrieval results and computation time in the deployment phase. The proposed multi-stage architecture is shown in Figure 9.1. The first stage consists of extracting the candidate datasets using a simple combined ranking scores from MCON and MultiEm-RGCN, as shown in the step M_1 from Figure 9.1. In this stage, recall is more important than precision to cover all possible relevant datasets. The irrelevant datasets can be discarded in the next stages. Then, for the second stage, the k_1 top ranked datasets from the first stage, denoted by C_1, are re-ranked using DSRMM to obtain a better set of relevant datasets. DSRMM is a supervised ranking method composed of convolutional filters and kernel pooling, and these operations are computationally efficient both in terms of time and memory. So, this second re-ranking

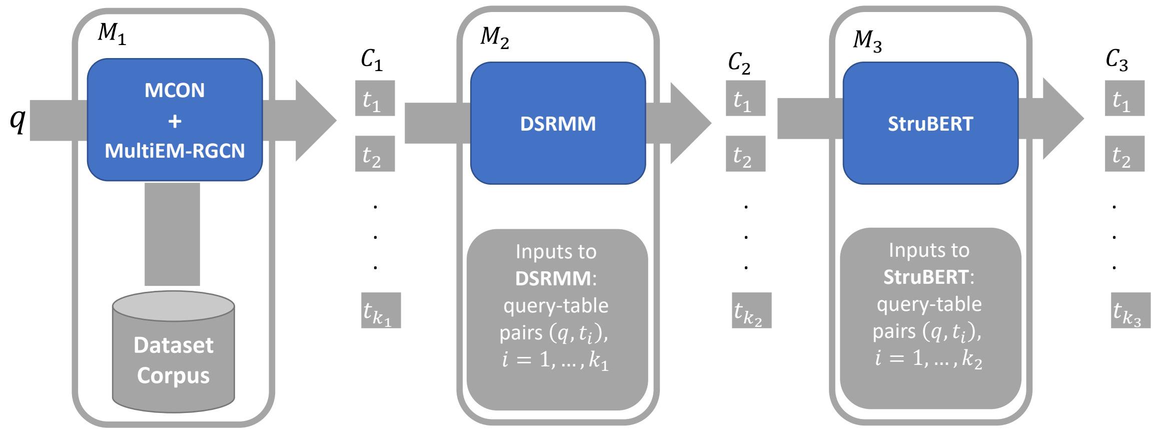

M_1
MCON + MultiEM-RGCN
Dataset Corpus
q
C_1
t_1
t_2
t_{k_1}
M_2
DSRMM
Inputs to DSRMM: query-table pairs (q, t_i), $i = 1, \ldots, k_1$
C_2
t_1
t_2
t_{k_2}
M_3
StruBERT
Inputs to StruBERT: query-table pairs (q, t_i), $i = 1, \ldots, k_2$
C_3
t_1
t_2
t_{k_3}

Figure 9.1: The multi-stage dataset search architecture. There are three stages M_1, M_2, and M_3. The inputs to the model are a user's query q and a dataset corpus. The models M_1, M_2, and M_3 return the set of datasets C_1, C_2, and C_3, respectively, where the set C_3 is returned to the user.

phase can boost the precision while preserving a high recall. Finally, in the third stage, StruBERT is used to rank the k_2 top datasets from the second phase, denoted by C_2, and return the final k_3 top ranked dataset to the user, denoted by C_3. So, StruBERT is trained to maximize the precision over the recall. This multi-stage model has the potential to reduce the number of datasets that should be ranked with StruBERT which is computationally expensive in terms of time and memory.

9.3 Introduction to dataset curation

Datasets are used by individuals or enterprises in multiple tasks. Retrieving relevant datasets to a user's query represents the first step in the process of generating useful knowledge from datasets. We proposed multiple methods to enhance dataset search so that users can find relevant datasets that satisfy their information needs. In order to extract useful knowledge from the retrieved datasets for downstream analytic tasks, a data cleaning step is necessary to improve data quality. This is related to data curation in general which consists of data integration such as schema matching and entity resolution, and data cleaning to identify and repair errors in the dataset. Data curation plays an important role in the automation of the data science process to extract valuable knowledge from datasets. Data curation is a time consuming task as data scientists often need to manually check for datasets for cleaning purposes. With the rapid increase of data volumes, automating data curation becomes a necessity to reduce time and cost of data analysis. Deep learning has been very effective in many tasks achieving human level performance in multiple fields such as natural language processing, speech recognition and computer vision. Therefore, deep learning has been leveraged in data curation, and has led to substantial improvement in multiple tasks such as data integration and entity resolution. In this dissertation, we focus on three specific problems. The first problem is related to the quality of schema labels in tabular datasets. This is a significant problem, because schema labels play an important role in dataset discovery [22, 23], schema matching [197, 282], data preparation and analysis [198], and integration of datasets [8, 221, 220]. The second problem is table similarity. This is also an important problem because predicting the semantic similarity between tables can be used in table classification and clustering [138, 280], and table fusion [78]. In addition, finding similar tables can be used to fuse cell values from matching tables for augmenting attributes [267] and auto-completion of data cells [294]. We previously showed that StruBERT [240] can be used to identify similar datasets. The third problem is entity matching [4] which consists of determining whether two records in two datasets refer to the same real-world entity. Entity matching is used for integrating different data sources. In the next two chapters, we propose new methods to solve both semantic labeling and entity matching.

Chapter 10

Context-aware Semantic Labeling using BERT

10.1 Introduction

Low quality schema labels lead to poor results in downstream tasks. We use the term schema label to refer to the information in the header row of a data table. Such tables typically have multiple rows and columns, and the first row is distinguished in that it names each of the columns. Sometimes these names are quite meaningful, and can be easily interpreted by a human reader; other times the names are quite opaque and the reader must hope external sources will provide clarity. In other cases, the schema labels were generated by an imperfect extraction process that leads to incorrect headers. In the worst situations, the header row is completely missing, and the only clues to the meaning of each column are the data values themselves, and if present, any additional metadata associated with the table. Even in the best case, where schema labels are present and clear, they can be extremely heterogeneous, leading to difficulties in automated processing [112, 105, 33, 166]. Existing methods generate schema labels solely on the basis of their content or data values, and thus ignore the contextual information of each column when predicting schema labels. The main research question to address in this chapter is how to formulate the semantic labeling to incorporate the contextual information of a given column when predicting schema labels.

To answer the research question, we present our formulation of the semantic labeling task that is based on the structured prediction, where the objective is to learn a mapping from a structured input (data table) to a structured label which consists of a sequence of predicted schema labels. In our formulation of semantic labeling, the structural dependencies between the data table and the predicted sequence of schema labels are leveraged using a sequential prediction of labels where each prediction is conditioned over the previous predictions. This is achieved by incorporating the already-predicted labels as contextual information for the current prediction of a given header. Our formulation reduces the ambiguity in predictions by incorporating contextual information that constraints the predicted sequence of schema labels similar to CRF constraints in POS tagging.

We propose a new *context-aware* semantic labeling method that incorporates both data values and column's context in order to infer the label. Our method presents a new setting for generating schema labels in which the input is a table with missing schema labels or headers, instead of the traditional setting that treats each column separately. An overview of the framework used in our method is described in Figure 10.1. Given a previously unseen table with missing headers, we sequentially predict schema labels, and incorporate the already-predicted labels as context for next header prediction within the same table. With our new setting, we formulate the semantic labeling as a structured prediction [56] problem where the objective is to learn a mapping from a structured input (data table) to a structured output (sequence of predicted semantic labels for data table's headers). Formulating the semantic labeling task as a structured prediction is motivated by the analogy with the image labeling task where the structured input consists of an image, and the structured outputs are the labels that are assigned to pixels or regions.

In summary, we make the following contributions:

- We propose a new context-aware semantic labeling approach. Our new formulation of semantic labeling is based on the structured prediction setting in which the input to our model is a data table with missing headers, and we sequentially generate schema labels for each data table.

- We demonstrate that by incorporating the predicted context of an attribute into the model, we can infer its context-aware schema label more accurately compared to methods that predict semantic labels using only data values.

- We are the first to adapt BERT into the semantic labeling task. In particular, we incorporate data values and the predicted context using a single BERT model that accepts two forms of inputs to make initial context-free and final context-aware predictions. Our proposed model is trained end-to-end for feature extraction and label prediction. This reduces human effort needed for semantic labeling.

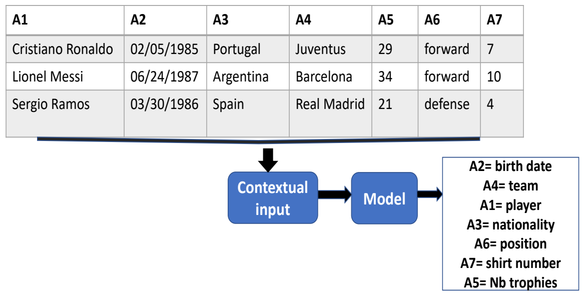
- We experiment on three datasets (public and internal data table corpus), and demonstrate that our new method outperforms the state-of-the-art baselines, and generalizes to table collections from multiple domains.

10.2 Problem statement

Our objective is to generate schema labels for table columns using data values and the predicted context in order to resolve the ambiguity problem in the prediction phase. For the rest of the dissertation, we use semantic labels and schema labels interchangeably.

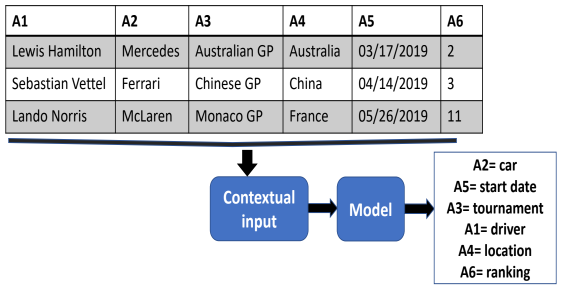
As we mentioned before, we use the multiclass classification setting to solve schema labeling. The training data consists of a table corpus of n data tables, $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$. Each table T_k has a set of m_k columns $A_1, A_2, \ldots, A_{m_k}$, where each column A_i has a schema label l_i (column's header), and a set of data values $V_i = \{v_1, v_2, \ldots, v_r\}$, where r is the number of rows in T_k. For the rest of the dissertation, we use m instead of m_k to denote the number of columns in $T_k \in \mathcal{T}$. The set of all possible schema labels is denoted by L.

Resolving ambiguity when predicting schema labels requires the whole table as input to the model, instead of only using independent column's values. Therefore, our setting consists of table inputs that have missing headers, and our objective is to predict schema labels for all columns of the input table. We denote our proposed model by $M = N \circ F$, where F is the feature extractor function (Contextual input block in Figure 10.1), and N is the classification layer (Model block in Figure 10.1). The input to M is a table T_k with missing schema labels, and the output of our model is a sequence of predicted schema labels $\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_m$. Our method learns both features and model simultaneously leading to significant reduction in human's effort spent in feature engineering.



A1	A2	A3	A4	A5	A6	A7
Cristiano Ronaldo	02/05/1985	Portugal	Juventus	29	forward	7
Lionel Messi	06/24/1987	Argentina	Barcelona	34	forward	10
Sergio Ramos	03/30/1986	Spain	Real Madrid	21	defense	4

(a) Predicting headers for a soccer table with true labels *player*, *birth date*, *nationality*, *team*, *Nb trophies*, *position*, *shirt number*



A1	A2	A3	A4	A5	A6
Lewis Hamilton	Mercedes	Australian GP	Australia	03/17/2019	2
Sebastian Vettel	Ferrari	Chinese GP	China	04/14/2019	3
Lando Norris	McLaren	Monaco GP	France	05/26/2019	11

(b) Predicting headers for a Formula 1 table with true labels *driver*, *car*, *tournament*, *location*, *start date*, *ranking*

Figure 10.1: The contextual input block extracts an embedding for each column using data values from all columns in a table. The model block maps the contextual input of each column into a probability distribution to predict the label of each column. The predicted labels for both data tables are shown in descending order of the prediction confidence. The contextual input solves the ambiguity of predictions. For example, A_3 from Table (a), and A_4 from Table (b) have data values related to class *country*. The context of A_3 using other headers, such as *player*, *team*, *position*, etc, and the context of A_4 that has *driver*, *car*, *tournament*, etc, can guide the model to predict *nationality* and *location* for A_3 (Table (a)) and A_4 (Table (b)), respectively.

10.3 Context Prediction for Semantic Labeling

In this section, we introduce our context-aware method for semantic labeling which is based on the structured prediction formulation. We formally define the contextual information of each column, which is combined with the column's data values to improve the performance of semantic labeling.

10.3.1 Column's context

The set of data values V_i for a given column A_i in a table T_k are not sufficient to have accurate schema label prediction. For example, in Figure 10.1, both columns *nationality* (A3 in the left table) and *location* (A4 in the right table) contain values from class *country*, but they refer to different labels. In this case, if we know that A_3 (in the left table) occurs in a table that contains *player*, *team*, *position*, and *birth date* attributes, hence it is more probable that A_3 is related to *nationality* rather than *location*. Therefore, we argue that the attributes $A_1, A_2, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m$ provides a rich contextual information for A_i, which we refer to as the **ground truth context** of A_i. However, as we explained in our setting, the input to our model is a table that has missing headers, which means that we cannot directly incorporate the context into our model.

To solve that, we propose incorporating the **predicted context** instead of the **ground truth context**. In other words, our model has two passes for predicting schema labels. During the first pass, given a table T_k with missing headers, only data values are used to make initial predictions for labels, denoted by $A'_1, A'_2, \ldots, A'_m$. The initial predictions are context-free, as they only capture data values.
For the second pass, we incorporate both data values V_i, and the **predicted context** $A'_1, A'_2, \ldots, A'_{i-1}, A'_{i+1}, \ldots, A'_m$ of A_i to make the final context-aware prediction, denoted by $\hat{A}_i$.

10.3.2 Semantic Labeling with BERT (SeLaB)

We incorporate data values and the predicted context of a given attribute using the contextualized language model BERT. So, for our proposed model $M = N \circ F$, denoted by *SeLaB*, F is equivalent to BERT with parameters θ, as we use the hidden state of [CLS] token from the last Transformer block to compute the embedding of the input sentence. N denotes the softmax layer with parameters W that produces the probability distribution of a given sequence over all schema labels from L.

As BERT is a model designed for use with linear text, one challenge is determining how best to provide table inputs to a BERT model. Our solution is to treat each column as a separate sentence that will be used in a different classification task. The two different forms of information we have for the column (its values and its predicted context) will be distinguished by [SEP] tokens. The general form of input to M for an attribute A_i, denoted by *contextual input*, is the sequence [CLS]+V_i+[SEP]+$context(A_i)$+[SEP], where $context(A_i)$ is the predicted context of A_i. For the first pass prediction, where $context(A_i)$ is missing, the input sequence form, denoted by *only values*, becomes [CLS]+V_i+[SEP]+[SEP]. As in information retrieval tasks [50, 156, 173, 271], applying BERT comes with the challenge of having longer sequences than the 512 tokens BERT allows. To satisfy the length limit requirement of BERT, we randomly select a subset of data values from V_i when the number of tokens in the *contextual input* or *only values* sequences exceeds the length limit. Next, we describe the training and testing phases.

Training phase

The steps of the training phase are shown in Algorithm 1. The inputs to the training phase are: table corpus $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ where semantic labels $l_1, l_2, \ldots, l_m$ are available for all attributes $A_1, A_2, \ldots, A_m$ of a table $T_k \in \mathcal{T}$, set of possible semantic labels L, and pre-trained BERT model as a feature extractor F. The compact notation of a table T_k, that is used in Algorithm 1, is $T_k = [[A_1, V_1], [A_2, V_2], \ldots, [A_m, V_m]]$.

The training phase has three stages. The first stage consists of predicting an initial label for each column using the *only values* input form as shown in Lines 3–7 of Algorithm 1. The output of the first stage is a sequence $A'_1, A'_2, \ldots, A'_m$ of initial predicted labels. During the second stage (Lines 8–12), we construct the predicted context $context(A_i)$ for each attribute A_i, which is the set of predicted labels $\{A'_j; j \in [1, m] \setminus \{i\}\}$. In order to avoid the true label leakage in $context(A_i)$, we remove l_i from $context(A_i)$ if $l_i \in context(A_i)$. We also remove duplicates from $context(A_i)$ as most data tables contain unique headers. The final stage (Lines 13–17) computes the context-aware predictions by using *contextual input* form. The output of M is the probability distribution $\hat{p}_i$ over all labels in L, for every $A_i \in T_k$. These probability distributions are used to calculate the cross-entropy loss, and to update the parameters of M as indicated in Lines 18–19. In addition to incorporating the context of column for schema labeling, our model accepts two forms of sequence inputs (*only values* and *contextual input*), which significantly reduces the number of parameters compared to the case where a separate model is needed to handle each type of input sequence.

In contrast to [199, 189] which have a pre-processing step to distinguish between string and numerical attributes, our BERT-based feature extractor F can process string and numerical texts by taking advantage of BERT tokenizers. In contrast to [33, 101] where the feature extraction and model building steps are decoupled, our model $M = N \circ F$ is trained end-to-end to jointly optimize the feature extractor F, and the classification layer N. SeLaB needs only BERT embeddings that is fine-tuned on a target table corpus to extract the feature vector of each column, and therefore generalizes to data tables from multiple domains. Unlike [28, 29] that integrate an external KB in semantic labeling with a strong assumption that the column's values match the KB entities, SeLaB needs only the standard BERT embedding that is fine-tuned to the target table corpus via end-to-end training. This model extracts

Algorithm 1 Training phase

```
1: Input: tables collection $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$, set of labels $L$.
2: for $T_k$ in $\mathcal{T}$ do    ▷ Schema labels of $T_k$, $l_1, l_2, \ldots, l_m$, are available
3:     for $[A_i, V_i]$ in $T_k$ do    ▷ First stage: Values based prediction
4:         input to BERT for $A_i$: $I_1(A_i) = [CLS] + V_i + [SEP] + [SEP]$
5:         compute values based probability, $p'_i = M(I_1(A_i))$
6:         $A'_i = \text{argmax}_{l \in L}\, p'_i[l]$
7:     end for
8:     for $[A_i, \_]$ in $T_k$ do    ▷ Second stage: Compute contexts of each attribute
9:         Context of $A_i$: $context(A_i) = A'_1 + A'_2 + \ldots + A'_{i-1} + A'_{i+1} + \ldots + A'_m$
10:        remove $l_i$ from $context(A_i)$ if $l_i \in context(A_i)$    ▷ avoid true label leakage in context
11:        remove duplicates from $context(A_i)$
12:    end for
13:    for $[A_i, V_i]$ in $T_k$ do    ▷ Third stage: compute final predictions
14:        input to BERT for $A_i$: $I_2(A_i) = [CLS] + V_i + [SEP] + context(A_i) + [SEP]$
15:        compute context-aware probability, $\hat{p_i} = M(I_2(A_i))$
16:        $\hat{A_i} = \text{argmax}_{l \in L}\, \hat{p_i}[l]$
17:    end for
18:    $loss(T_k) = CrossEntropy([\hat{p_1}, \hat{p_2}, \ldots, \hat{p_m}], [l_1, l_2, \ldots, l_m])$
19:    update $M$ parameters $(\theta, W)$ by minimizing $loss(T_k)$
20: end for
21: Output: A Trained context-aware model $M$
```

the feature of each column, and therefore generalizes to data tables from multiple domains.

Testing phase

The steps of the testing phase are shown in Algorithm 2. The inputs to the testing phase are: a testing table T_k that has missing headers ($l_1, l_2, \ldots, l_m$, are not available), set of possible semantic labels L, trained model M, and two parameters *unique_headers* and top_k that we will describe later.

The testing phase has three stages. The first and second stages are similar to the training phase, where initial predictions are computed using *only values* input form, and then used to produce the context of each attribute. During the third stage, the final predicted labels for the testing data table are generated sequentially. For a given table T_k, initially all schema labels are missing.

Given that the prediction is done sequentially, m passes are needed to obtain a predicted label for each column in T_k. For the j-th pass, the set of decided columns has $j-1$ labeled headers, and $m-j+1$ columns in T_k, denoted by S_j, are still undecided (missing the predicted labels). We predict the probability distribution $\hat{p}_i$, and a schema label $\hat{A}_i$ for each column $A_i \in S_j$ using our model M with *contextual input* sequence similar to lines 15 and 16 of training phase, respectively. The confidence of prediction for $A_i \in S_j$ is given by $p^i_{max} = \max_{l \in L} \hat{p}_i[l]$. The *unique_headers* is a Boolean variable that is set to *True* to force the unique headers constraint for a given table. When predicting duplicate headers is allowed, the column A_h, that we choose to predict from S_j in the j-th pass, is given by $A_h = \text{argmax}_{A_i \in S_j} p^i_{max}$, and we set the label of A_h to $\hat{A}_h$ as shown in Lines 15–16.

Algorithm 2 Testing phase

```
1: Input: testing table $T_k$, set of headers labels $L$, trained $M$ model, Boolean: unique_headers
   and $top_k$.
2: %First stage: Values based prediction
3: %Second stage: Compute contexts of each attribute
4: $S_1 = T_k$                                          ▷ initial set of undecided columns
5: predicted_attributes = $\emptyset$                   ▷ initial set of predicted schema labels
6: for $j \in 1, 2, \ldots, m$ do                       ▷ Third stage: compute final predictions
7:     for $A_i \in S_j$ do
8:         input to BERT for $A_i$: $I_2(A_i) = [CLS] + V_i + [SEP] + context(A_i) + [SEP]$
9:         compute context-aware probability, $\hat{p}_i = M(I_2(A_i))$
10:        $\hat{A}_i = \text{argmax}_{l \in L} \hat{p}_i[l]$,    $p^i_{max} = \max_{l \in L} \hat{p}_i[l]$
11:    end for
12:    if unique_headers then
13:        $A_h, chosen\_label = UniqueHeaders(\{\hat{p}_i, A_i \in S_j\}, top_k, predicted\_attributes)$
14:        $\hat{A}_h = chosen\_label$
15:    else
16:        $A_h = \text{argmax}_{A_i \in S_j} p^i_{max}$
17:    end if
18:    $predicted\_attributes.append(\hat{A}_h)$
19:    $S_{j+1} = S_j \setminus \{A_h\}$
20:    replace $A'_h$ by $\hat{A}_h$ in $context(A_i)$, $A_i \in S_{j+1}$  ▷ update contexts with new predicted attribute
   $\hat{A}_h$
21: end for
22: Output: A label for each header in the testing table.
```

On the other hand, when unique headers constraint is required for a given data table, we propose a routine, called *UniqueHeaders*, that resolves the duplicate headers problem. The inputs to this routine are: the probability distributions $\hat{p}_i$ for $A_i \in S_j$, top_k which denotes the number of top confidences per attribute that are used to find the label, and the set *predicted_attributes* that contains the $j-1$ semantic labels that are already assigned to $j-1$ columns of T_k. The objective of the function *UniqueHeaders* is to find the label *chosen_label* with the highest confidence value, with respect to the unique headers constraint that requires $chosen\_label \notin predicted\_attributes$. For time complexity efficiency, we limit the depth of search by choosing $top_k << |L|$. By limiting the depth of search, *UniqueHeaders* can produce a duplicate header. In this case, we use a heuristic that returns the label that corresponds to the maximum confidence score. $UniqueHeaders$ is called in Lines 12–14 of Algorithm 2.

We remove the chosen column A_h from S_j to obtain S_{j+1} (the columns of T_k that are still missing labels after the j-th pass), and we add $\hat{A}_h$ to the set of predicted schema labels *predicted_attributes* (Lines 18–19). We finish the j-th pass by updating $context(A_i)$, where $A_i \in S_{j+1}$, using the predicted label $\hat{A}_h$ from the j-th pass as shown in Line 20. The objective of the context update is to replace the *only values* prediction A'_h by the *contextual input* inferred label $\hat{A}_h$ in $context(A_i)$ for $A_i \in S_{j+1}$, as the latter is more accurate than the former. For the j-th pass, we select the best schema label from $m - j + 1$ predicted labels. So, for m passes, the total number of predictions, that are needed to infer the schema labels of T_k, is $m \times (m-1)/2$. The increase in the number of predictions is justified by the sequential nature of the testing phase where the context of each undecided attribute is updated in each pass, and the most confident prediction is selected.

10.4 Evaluation

10.4.1 Baselines

We compare the performance of our proposed model with feature-based baselines [33, 101], and a variation of our model where only data values (without context) are used. We describe the five categories of features that are extracted from the data values of each column. There are five categories of features as shown in Table 10.1. Four categories are previously used in feature-based methods: global statistics [33, 101], character distributions [33, 101], word embeddings [101], and paragraph embeddings [101]. To obtain more fine-grained embedding, we also propose a character-based generative model to produce character embeddings for each value.

Global statistics

We combine the global statistics from [33] and [101] into one category that has 17 unique features with different dimensions as shown in Table 10.2. When concatenating the global statistics features, the dimension of the resulting feature vector is 52.

Table 10.1: Features categories used in feature based methods (total dimension equals to 2213)

ID	Category type	Dimension
1	Global statistics	52
2	Character distributions	960
3	Character embeddings	400
4	Word embeddings	401
5	Paragraph embeddings	400

Table 10.2: Global statistics features [33, 101] with a total dimension equals to 52

ID	Length	Description
1	1	minimum value in column's content
2	1	maximum value in column's content
3	1	average value of column's content
4	1	standard deviation value of column's content
5	1	percentage of numeric cells in column's content
6	20	content histogram
7	1	number of values in a column
8	1	column entropy
9	1	percentage of unique content
10	1	percentage of values with numeric characters
11	1	percentage of values with alphabetical characters
12	2	mean and standard deviation of the number of numerical characters in column's content
13	2	mean and standard deviation of the number of alphabetical characters in column's content
14	2	mean and standard deviation of the number of special characters in column's content
15	2	mean and standard deviation of the number of words in column's content
16	4	None values statistics: count, percentage, has some None values(Boolean), has Only None values (Boolean)
17	10	length of values statistics: any nonzero length (Boolean), all nonzero length (Boolean), sum, min, max, variance, median, mode, kurtosis, skewness

Character distributions

The distribution of characters is computed for each column using 96 ASCII-printable characters. For each character, 10 statistics (any, all, mean, variance, min, max, median, sum, kurtosis [1], skewness [2]) are calculated based on the count of each character in a value from the set of data values for the input column. Concatenating character distribution for all characters results in a feature vector with dimension 960.

Character embeddings

We train a character-level language model [226] to generate values from the table corpus $\mathcal{T}$. Our generative model has two character-based LSTM layers, and it is trained on the next character generation task. Given a value v_k, which is a sequence of characters, our generative model produces a hidden state for each character, and we use the hidden state from the second LSTM layer of the last character as the

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kurtosis.html
[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.skew.html

character embedding of v_k. Then as in [101], we compute the mean, mode, median and variance of character embeddings across all values $v_k \in V_i$ in a column A_i. Given that the dimension of character embedding is 100, and we have 4 statistics, the resulting feature vector has dimension 400.

Word embeddings

The pre-trained word embedding, Glove[3] [185], is used to compute embedding for each value $v_k \in V_i$. Then, as in character embedding, 4 statistics are computed for V_i. The dimension of embedding is 100, so that after computing statistics, the dimension of the concatenated feature vector equals 400. As in [101], an additional binary feature is appended to the final word embedding feature vector, and it indicates if there is at least one value from V_i that belongs to Glove vocabulary.

The use of pre-trained embedding is suitable for table collections, such as WikiTables, that have common values with the vocabulary of the pre-trained embedding. This is not the case for log tables from network equipment, where the number of out of vocabulary (OOV) tokens is large, and this leads to poor performance for pre-trained word embedding. To solve this problem, we train a word embedding on the table corpus $\mathcal{T}$, where the sentences are rows and columns from $T_k \in \mathcal{T}$. Instead of using Glove, we train a fastText [12] model to produce word embeddings. The use of character-level n-grams in fastText allows word embeddings to be created even for terms that have not been seen before, and reduces the negative effect of OOV tokens. Given that having long strings is common for log data, we use BERT tokenizer to preprocess values before training fastText embeddings.

Paragraph embeddings

Each column A_i can be seen as a paragraph that contains the set of values $V_i = \{v_1, v_2, \ldots, v_r\}$. The paragraph embedding, that is based on the distributed bag of words [131], is trained to map each column into an embedding with a dimension equals to 400.

[3]https://nlp.stanford.edu/projects/glove/

Sherlock

Hulsebos et al. [101] uses global statistics, character distributions, word embeddings, and paragraph embeddings with a multi-input neural networks architecture.

All features

This baseline extends Sherlock [101] features by adding our character embeddings to cover three different levels of embeddings (character, word, and paragraph).

BERT with only values

This baseline can be seen as a variation of our proposed method SeLaB, where only data values are used to predict schema labels. So, for training, the first stage predictions are used to update the parameters of the model. For testing, the first stage predictions are used to evaluate the performance of the model. The input data values sequence to the model has *only values* input form.

10.4.2 Experimental Setup

In our proposed model, we use the BERT-base-uncased for the feature extractor F. For each column, we shuffle data values and randomly select a subset of values to reduce the complexity of the model. Given that the majority of log tables have more rows than WikiTables and the combined data, we randomly choose 200 values for each column in a given log table, and 100 values for columns from WikiTables and the combined data. In general, shuffling the predicted context can reduce overfitting. For the WikiTables collection, the majority of tables that have both attributes *home team* and *away team* (these attributes have similar data values), report *home team* column before *away team* column (same for *birth date* which occurs before *death date*) for a left-to-right sequential order. In this case, we can resolve the ambiguity of predicting *home team* and *away team* by keeping the sequential order of the predicted context. We train our model for 10 epochs, and we set top_k of the $UniqueHeaders$ routine to 5.

The model is implemented using PyTorch [184], with Nvidia GeForce GTX 1080 Ti. Our implementation is based on BERT codes from an open source repository[4].

[4]https://github.com/huggingface/transformers

Method Name	Macro-P	Macro-R	Macro-F	Micro-F	MRR
Global statistics	7×10^{-3}	10^{-2}	6×10^{-3}	0.17	0.28
Character distributions	0.14	0.10	0.10	0.38	0.48
Character embeddings	0.38	0.33	0.33	0.54	0.64
Word embeddings	0.18	0.16	0.15	0.47	0.58
Paragraph embeddings	0.23	0.19	0.19	0.43	0.54
Sherlock	0.45	0.45	0.42	0.66	0.75
All features	0.54	0.47	0.47	0.66	0.75
BERT (only values)	0.46	0.45	0.43	0.64	0.73
SeLaB w/o unique headers	**0.55**	0.52	0.50	**0.72**	**0.80**
SeLaB	**0.55**	**0.53**	**0.51**	**0.72**	**0.80**

(a) WikiTables

Method Name	Macro-P	Macro-R	Macro-F	Micro-F	MRR
Global statistics	0.16	0.16	0.15	0.54	0.64
Character distributions	0.40	0.42	0.40	0.68	0.71
Character embeddings	0.44	0.44	0.43	0.69	0.71
Word embeddings	0.35	0.34	0.33	0.65	0.69
Paragraph embeddings	0.31	0.29	0.29	0.61	0.67
Sherlock	0.42	0.42	0.41	0.67	0.73
All features	0.50	0.49	0.49	0.71	0.72
BERT (only values)	0.48	0.49	0.47	0.73	0.76
SeLaB w/o unique headers	0.61	0.62	0.60	**0.81**	**0.81**
SeLaB	**0.62**	**0.63**	**0.61**	**0.81**	**0.81**

(b) Network log tables

Table 10.3: Semantic labeling results

We use the Adam [120] optimizer for gradient descent to minimize the cross-entropy loss function and update the weights of our model. For the baselines, BERT with only values is also trained for 10 epochs. For the feature-based baselines (except for Sherlock), random forest from the scikit-learn implementation [5] is trained for prediction.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

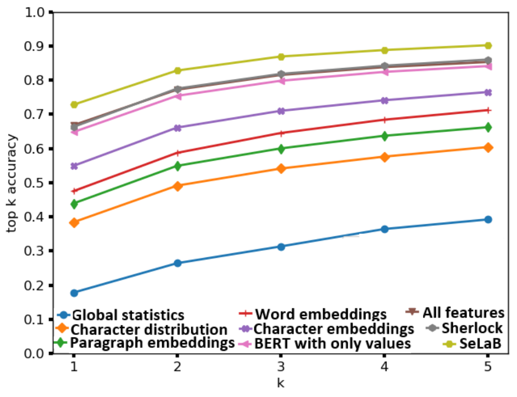
10.4.3 Experimental results

We evaluate the performance of SeLaB and baselines on the schema labeling task using macro-averaged and micro-averaged precision (P), recall (R) and F-score of predictions on the testing set. In the multiclass classification problem, the micro-average precision, recall and F-score are the same, so we only report the Micro-F score. We also report the Mean Reciprocal Rank (MRR) [144], as the rank of the true class is an important measure for evaluation. We calculate the top-k accuracy that shows the fraction of testing samples where the true label is within the top k predicted confidence scores. We report results for our method and baselines using five-fold cross validation.

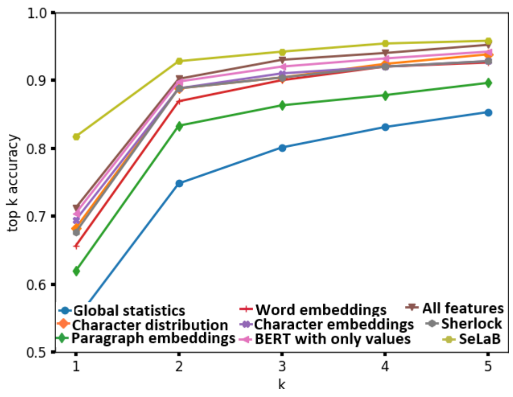
Semantic labeling results

Table 10.3(a) shows the performance of different approaches on the WikiTables collection. We show that our proposed method SeLaB outperforms the baselines for all evaluation metrics. By formulating the semantic labeling as a structured prediction problem, we were able to incorporate the contextual information of each attribute into our model to solve the ambiguity in predictions, and this leads to an increase in evaluation metrics compared to baselines that generate schema labels solely on the basis of data values. If we simply use BERT without structured prediction, it is comparable to the state-of-the-art Sherlock (Macro-F of 0.43 vs. 0.42, Micro-F of 0.64 vs. 0.66), but when we add in the predicted context, both Macro-F and Micro-F improve by 0.08. Among all the five categories of features (global statistics, character distributions, character embeddings, word embeddings, paragraph embeddings) that are used in the feature-based approaches, our character embeddings feature achieves higher performance for all evaluation metrics. So, a generative model with a character granularity is able to capture the distributions of data values drawn from different attributes. For WikiTables, the standard deviation of SeLaB in all evaluation metrics is less than 0.08.

Figure 10.2(a) shows the top-k accuracy results on WikiTables where our method SeLaB outperforms the baselines. "Bert with only values", Sherlock, and "All features" baselines have close performances. So, the BERT-based embedding, which is trained by using only data values, is as good as the hand-crafted features. While extracting the hand-crafted features requires significant human effort to compute the



(a) WikiTables collection



(b) Network log tables collection

Figure 10.2: Top-k accuracy results

global statistics, character distributions and three types of embeddings (character, word, and paragraph), BERT embeddings are trained jointly with the classification layer with minimal preprocessing which reduces the human effort. Among all the five categories of features (global statistics, character distributions, character embeddings, word embeddings, paragraph embeddings) that are used in the feature-based approaches, our character embeddings feature achieves higher performance for top-*k* accuracy results. So, a generative model with a character granularity is able to capture the distributions of data values drawn from different variables or attributes.

Table 10.3(b) and Figure 10.2(b) show the performance of the different approaches on the Log tables from network equipment. Interestingly, Sherlock has a worse Macro-F and MRR on network log tables than on WikiTables, while SeLaB and BERT (only values) both improve on all metrics with this alternative dataset. Consistent with WikiTables, our results on the log tables corpus show the importance of a column's context in improving the semantic labels prediction, especially for top-1 accuracy as shown in Figure 10.2(b), where SeLaB was the only system to achieve a value over 0.8. The top-5 accuracies for SeLaB, "Bert with only values", and "All features" are similar which indicates that the ambiguity in semantic labeling occurs mainly when predicting exact schema labels. For Log tables, the standard deviation of SeLaB in all evaluation metrics is less than 0.09. Semantic labeling results on WikiTables and Log tables show that SeLaB achieves significant improvements in the evaluation metrics of two data table collections from different domains, which supports the generalization characteristic of our proposed method.

We summarize the evaluation metrics results from the third dataset in Table

10.4. Consistent with WikiTables and Log tables, the ambiguity in the schema label prediction is more dominant when predicting exact labels, which is related to the top-1 accuracy. SeLaB outperforms baselines mainly in top-1 accuracy by taking advantage of both data values and predicted context. For the combined data, the standard deviation of SeLaB in all evaluation metrics is less than 0.06.

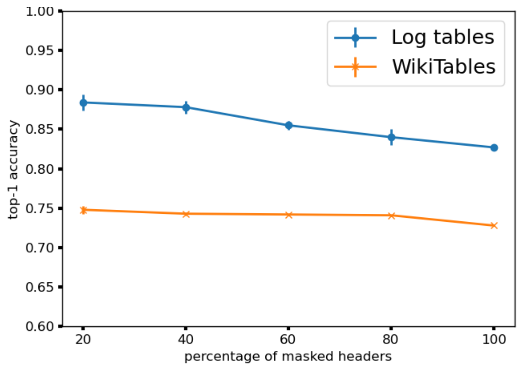
Method Name	Macro-P	Macro-R	Macro-F	Micro-F	MRR	Top-1	Top-2	Top-3	Top-4	Top-5
Global statistics	0.05	0.08	0.05	0.16	0.30	0.16	0.29	0.36	0.41	0.44
Character distributions	0.32	0.33	0.30	0.43	0.54	0.43	0.53	0.61	0.63	0.66
Character embeddings	0.32	0.33	0.30	0.43	0.55	0.43	0.56	0.62	0.67	0.70
Word embeddings	0.29	0.31	0.28	0.39	0.50	0.39	0.50	0.55	0.57	0.61
Paragraph embeddings	0.32	0.33	0.30	0.47	0.57	0.47	0.56	0.60	0.65	0.67
Sherlock	0.38	0.38	0.36	0.49	0.60	0.49	0.61	0.67	0.71	0.74
All features	0.43	0.42	0.40	0.54	0.65	0.54	0.66	**0.72**	**0.75**	**0.78**
BERT (only values)	0.38	0.41	0.37	0.52	0.62	0.52	0.64	0.68	0.71	0.72
SeLaB	**0.45**	**0.49**	**0.45**	**0.59**	**0.66**	**0.59**	**0.67**	**0.72**	**0.75**	0.77

Table 10.4: Semantic labeling results for the combined data

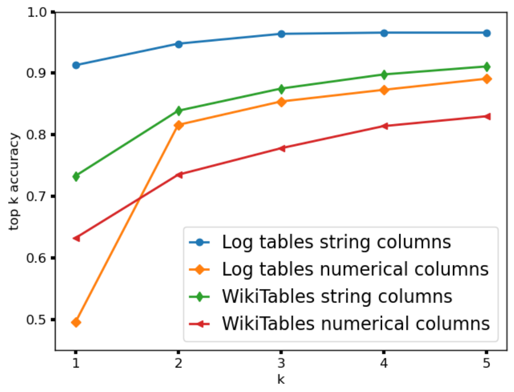
Masked headers

So far, we have discussed the performance of SeLaB where all semantic labels are missing for a given data table. This can be seen as an extreme case. A common scenario for table extraction is to have a percentage of missing or false headers. To better understand how SeLab deals with such tables, we randomly mask a percentage of headers from tables in the testing set, and we report the top-1 accuracy function of the percentage of masked headers as shown in Figure 10.3(a). For each table, we run the masked headers experiment five times with randomly selected headers, and we compute the mean and standard deviation (std) of the top-1 accuracy for each percentage of masked headers. 100% masked headers means that all labels are missing which is the most difficult task (and is equivalent to the previous experiment).

As reported in Figure 10.3(a), the maximum std (vertical bar) is 0.01 for Log tables and 0.005 for WikiTables. Figure 10.3(a) shows that when the percentage of masked headers decreases, there is an increase in the mean of top-1 accuracy for predicted masked headers. This means that the attribute's context is more accurate given that the labels of the non-masked headers are groundtruth labels. However, comparing the fully predicted context in 100% masked headers with the partially



(a) top-1 accuracy for masked headers



(b) top-k accuracy for string and numeric

Figure 10.3: Top-k accuracy for masked headers and string/numeric headers

predicted context in 20% masked headers, there is only a small increase in the top-1 accuracy (0.057 for Log tables and 0.020 for WikiTables) which indicates that the predicted context in the extreme case is as good as the groundtruth context.

String vs Numeric columns analysis

We evaluate the performance of SeLaB on two categories of columns which are numeric and string. As shown in Figure 10.3(b), we report the top-k accuracy of numerical and string columns for Log tables and WikiTables. In both datasets, semantic labeling of string columns outperforms numerical columns. So, generating an exact schema label for numerical data values is more ambiguous than string values, as numerical columns contain similar values. In particular, the increase of the top-2 accuracy compared to the top-1 accuracy for numeric columns in network log tables is justified by the presence of two semantic labels (*line_number* and *row_index*) that are used interchangeably. Thus, the set of the top-2 predictions contains both labels, and this leads to a significant increase in the top-2 accuracy compared to top-1 accuracy.

Predicted label examples

To better understand how SeLaB works, we show examples of predicted schema labels from WikiTables testing set in Table 10.5. Each row corresponds to a testing table where we show the ground truth attributes, first stage predictions, and the final predicted schema labels. For example, for the first row, there are three wrong predictions (*year* instead of *season*, *division* instead of *section*, and *finish* instead

Table attributes	First stage predictions	Third stage predictions
season, level, division, section, position	year, level, division, division, finish	**season**, level, division, **section**, **position**
year, title, developer, publisher, setting, platform	year, title, developer, developer, setting, system	year, title, developer, **publisher**, setting, system
pos, rider, bike, pts	rank, rider, bike, pts	**pos**, rider, bike, pts
pos, class, no, team, drivers, car, laps, qual pos	pos, group, rank, driver, driver, car, deaths, rank	pos, **class**, **no**, entrant, **drivers**, car, **laps**, grid
senator, party, years, term, electoral history	representative, party, years, wins, electoral history	representative, party, years, **term**, electoral history
date, time, home team, away team	date, time, home team, home team	date, time, home team, **away team**
site, location, year, description	site, province, year, description	name, **location**, year, description
land area, latitude, longitude	area, latitude, geographic coordinate system	**land area**, latitude, **longitude**
title, director, cast, genre, notes	film, role, cast, genre, notes	role, **director**, cast, genre, notes
county, location, exit number, destinations, notes	county, location, notes, notes, notes	county, location, exit, **destinations**, notes

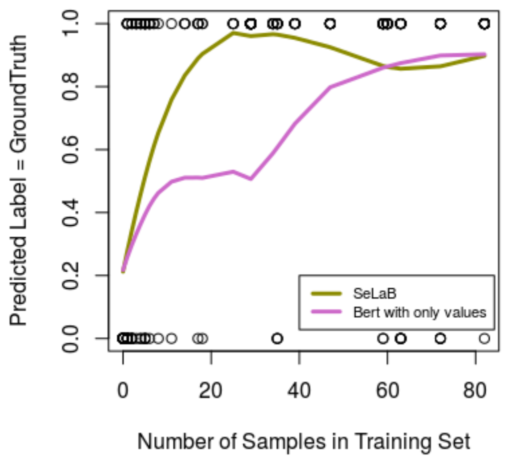
Table 10.5: Examples of predicted schema labels from Wikitables testing set

of *position*) from first stage predictions which are based only on data values. After incorporating context for each attribute, SeLaB updates the predicted label for each column, and the new context-aware semantic labels that match the ground truth labels are shown in bold in Table 10.5. For the first example, we obtain the third stage predictions which are identical to the ground truth labels after three corrections from context-aware representation for each column. For the sixth row, the table's attributes contain *home team* and *away team*. Both attributes are predicted *home team* after first stage predictions. SeLaB learns that *away team* occurs with *home team* so that the predictions are corrected after the third stage.

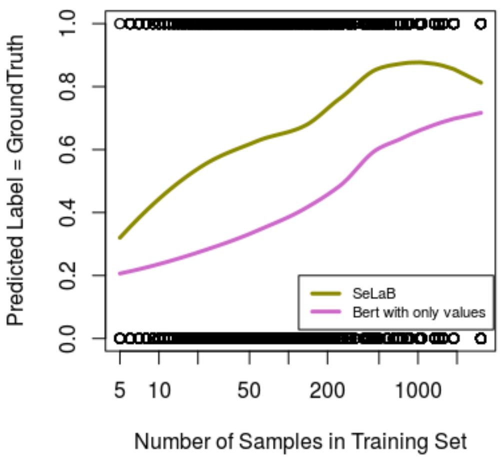
Effect of the Number of Training Samples

To understand how the number of training samples for each semantic label affects the accuracy of SeLaB predictions, we show in Figure 10.4 the indicator values of correct testing predictions per label against the number of samples for each label in the training set as black circles (i.e. 1 indicates the predicted column header is the same as the ground truth).

Local smoothing [44] was applied to obtain the average accuracy curve for SeLaB predictions (yellow line). As a reference, we also added a similar local smoothing curve representing the accuracy curve from predictions obtained using "Bert with only values" (pink line). Figure 10.4 demonstrates that in general, as the number of samples for each label in the training set increases, both SeLaB and "Bert with only values" perform better. However, SeLaB appears to perform much better when there is a sufficient number of samples per label (e.g. more than 20 instances). There is a slight dip for the SeLab curve for those columns when the corresponding number of samples in the training set becomes the largest. Upon close inspection, we noticed that the most frequent semantic labels in the training set are generic labels. For



(a) Log tables



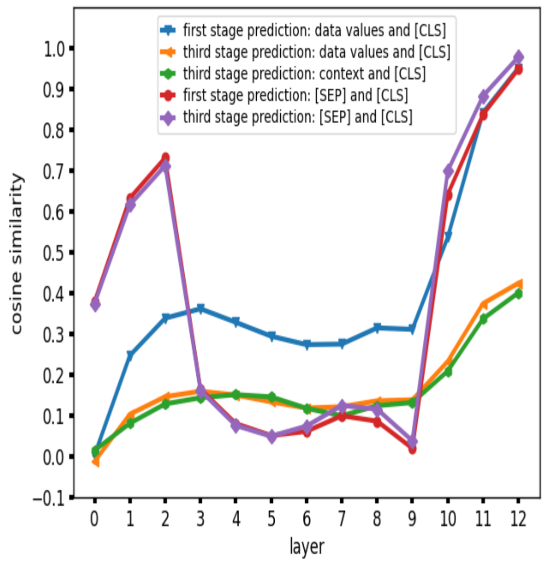
(b) WikiTables

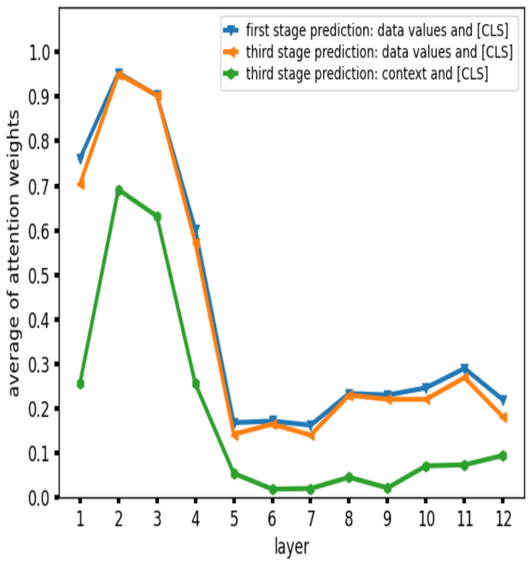
Figure 10.4: Accuracy vs. Number of training samples per label

example, in Wikitables, the column header with the largest number of instances is in fact a generic label *name* (with 3064 instances in training data). In the testing set, SeLaB sometimes predicts *name* as *player*, *swimmer*, *representative*, etc., depending on the context of the table, thus potentially yielding more appropriate header names than the ground truth.

Comparison of SeLaB embedding and attention weights from multiple layers

SeLaB takes the final hidden state of the [CLS] token as the representation of the whole input sequence, and we are interested in discovering the tokens that have similar embedding to [CLS] token, and the tokens that have the largest impact on the [CLS]. To better understand the embedding of the [CLS] token in SeLaB, we compute the cosine similarity between the embedding of the [CLS] token and the embedding of multiple tokens in both *only values* and *contextual input* forms. Figure 10.5(a) illustrates multiple cosine similarities that are computed in multiple layers. There are 13 layers in Figure 10.5(a) where layer 0 is the embedding layer in BERT, and the next 12 layers are the Transformer blocks. The [SEP] delimiter is an important token in BERT that marks the end of a sequence. So, we start by showing the cosine





(a) Average of cosine similarity between the embedding of [CLS] token and different tokens

(b) Average of attention weights between the embedding of [CLS] token and different tokens

Figure 10.5: Analysis of embeddings and attention heads using the combined dataset

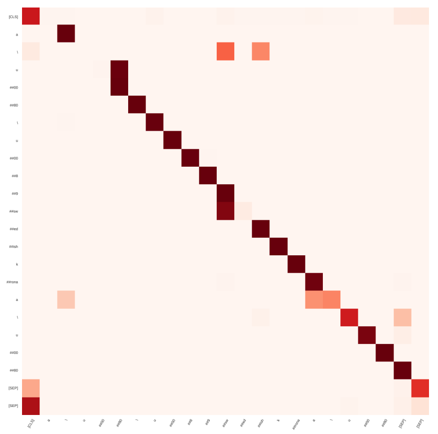
similarity between the embedding of [CLS] and [SEP] tokens in both first and third stage predictions. We have two [SEP] tokens in both *only values* and *contextual input* forms, and we plot the average of cosine similarities between [SEP] tokens and the [CLS] token in multiple layers. As shown in Figure 10.5(a), the cosine similarity between the embedding of [SEP] and [CLS] is large, especially in deep layers starting from layer 10, where these layers capture the interactions between the high-level representations of multiple tokens. The high cosine similarity between the embedding of [CLS] and [SEP] tokens in both first and third stage prediction can be explained by the convergence of the embeddings of special tokens in BERT ([CLS] and [SEP]), and suggest that at the end of the fine-tuning phase, these tokens have similar features.

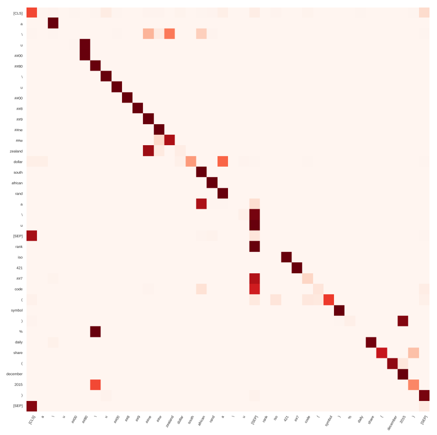
To justify that SeLaB updates the predicted schema label in the third stage using both data values and the attribute's context, we show the cosine similarity between the [CLS] token and both the data values and attribute's context in the first and third stage prediction. For both *only values* and *contextual input* forms, we extract the embeddings that correspond to the values tokens and average them to obtain the data values embedding for the first and third stage prediction, respectively. For the *contextual input* form, we extract the embeddings of the context tokens, and average them to obtain the context embedding of the third stage prediction. Comparing the cosine similarity in all layers between the data values and [CLS] in the first stage prediction with the cosine similarity between the data values and [CLS] in the third stage prediction, we observe a drop in the cosine similarity for the third stage prediction. This phenomenon is expected because in the third stage prediction, SeLaB attends to context tokens in addition to data values tokens as shown by the curve from Figure 10.5(a) that illustrates the cosine similarity between the context tokens and [CLS] in the third stage prediction. SeLaB accords comparable importance to data values and context when inferring schema labels in the third stage prediction which justifies the importance of both fields.

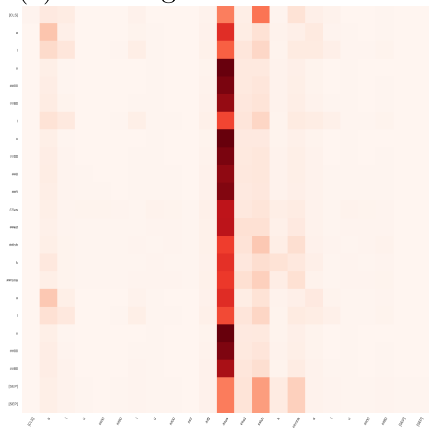
In addition, to better understand the information that is captured by the [CLS] token in SeLaB, we compute the average of attention weights from the top 20 tokens of each field (data values and predicted context) that attend or are attended by the [CLS] token in all attention heads of a given layer. Figure 10.5(b) illustrates the average of attention weights in multiple layers (12 Transformer blocks) from both first and third stage predictions. We observe similar patterns in all three curves where deeper layers tend to focus on specific tokens which leads to a lower average of attention weights compared to first layers. We also observe that the average of attention weights in the context field is lower than the average of attention weights of data values in all layers. This can be explained by the presence of more variety in data values (data tables can have different sets of values for the same attribute) compared to the context field, where in some cases the presence of one specific token in the context field can be sufficient to resolve the ambiguity in predicting the schema label.

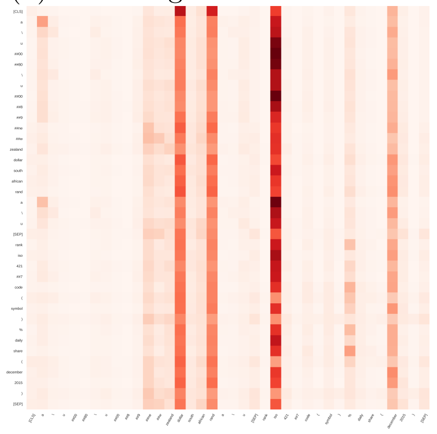
Comparison of SeLaB attentions from multiple layers (first example)

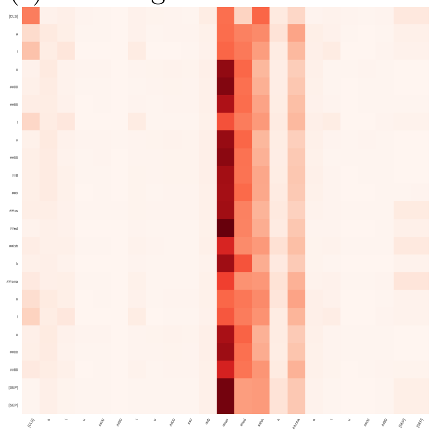
The Transformer blocks are the main components that have led to state-of-the-art results in multiple tasks with their ability to capture long-range dependencies better than the recurrent architectures. We show the attention heads of Transformer blocks in SeLaB to better understand the prediction mechanism in the testing phase, and to explain how SeLaB infers schema labels in both the first and third stages of the testing phase. Figure 10.6 illustrates attention heads from multiple layers where the first column represents first stage attention heads, and the second column represents third stage attention heads. The data table, that is used in this example, contains


(a) first stage: 1st attention head at layer 3


(b) third stage: 1st attention head at layer 3


(c) first stage: 6th attention head at layer 12


(d) third stage: 6th head at layer 12


(e) first stage: 11th head at layer 12

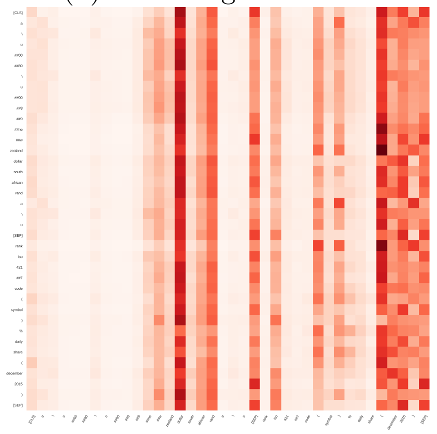

(f) third stage: 11th head at layer 12

Figure 10.6: Comparison of attention heads between first and third stage prediction. Attention weights with larger absolute values have darker colors.

the attributes: *rank*, *currency*, *iso 4217 code (symbol)*, and *% daily share (december 2015)*. After the first stage prediction, the schema label *currency* is mislabeled by

SeLaB. To explain the reason for misclassifying the *currency* attribute in the first stage and correctly classifying this attribute in the third stage, we show the attention heads that are related to the prediction of the *currency* attribute.

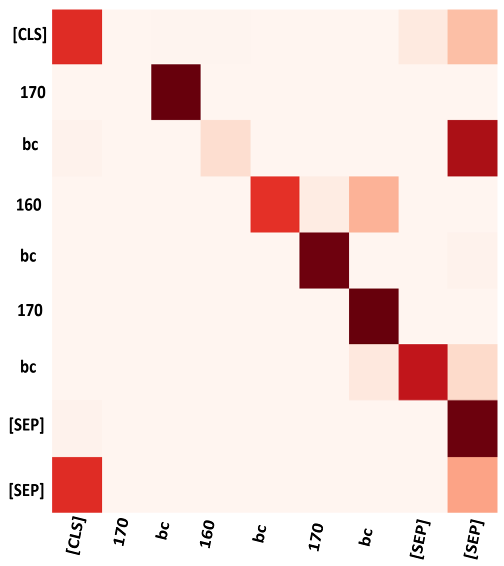
Figures 10.6(a), (c), and (e) illustrate multiple forms of attention heads after the first stage prediction. In general, there are multiple attention heads as categorized by Kovaleva et al. [125]. In the first stage prediction, SeLaB focuses on a set of values tokens (##sw, ##ed, ##ish, ##rona) which are not enough to infer the correct schema label. After incorporating the context of the attribute *currency* to infer the third stage prediction, we show similar attention heads to the first stage prediction. Figures 10.6(b), (d), and (f) illustrate multiple attention heads after the third stage prediction. SeLaB focuses on both the data values and the context fields, where the value *dollar*, and the context tokens (*iso*, *december*, *2015*) are given more attention and therefore contribute the most to the correct updated schema label in the third stage prediction.

Comparison of SeLaB attentions from multiple layers (second example)

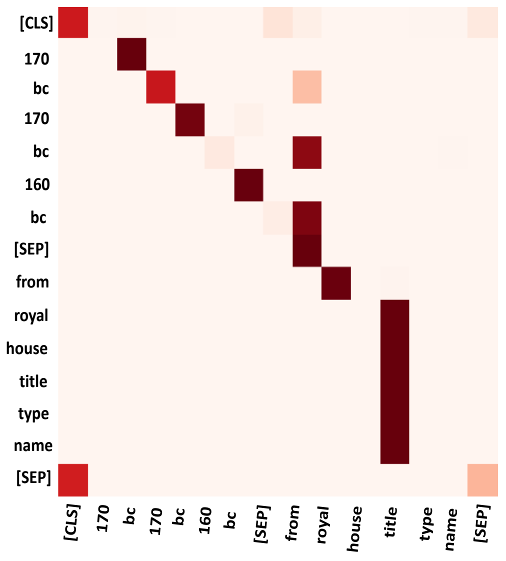
We show the attention maps of a data table with the attributes: *type*, *name*, *title*, *royal house*, *from*, and *to*. Both attributes *from* and *to* contain similar values such as 185 BC, 170 BC, 160 BC, etc. These two attributes are predicted *from* after the first stage prediction, then the wrong prediction is corrected after the third stage prediction, and we obtain a correct sequence of predicted schema labels. To better understand how SeLaB updates its prediction, we compare multiple attention heads from multiple layers in both first and third stage prediction.

Figure 10.7(a) shows the 1st attention head at layer 3 from the first stage prediction, which is mostly diagonal and each token attends to itself. This layer forms low-level features. Figure 10.7(c) illustrates the 11th attention head at layer 12 from the first stage prediction. This attention head has a grid form, with a focus on the token *bc*.

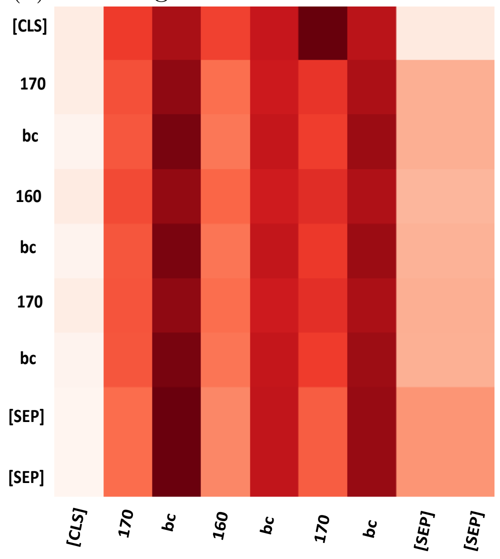
After incorporating the context of the attribute *to* to infer the third stage prediction, we show similar attention heads to the first stage prediction. Figure 10.7(b) illustrates the 1st attention head at layer 3 from the third stage prediction. We observe that two tokens (*from* and *title*) from the context field are given more attention, which indicates that SeLaB starts to update its prediction with the presence of the



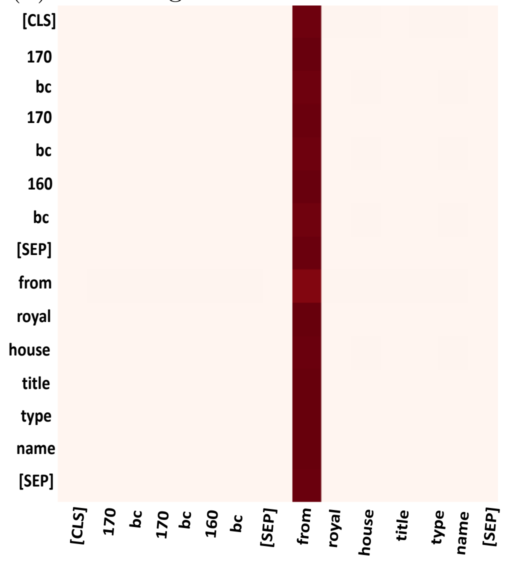
(a) first stage: 1st attention head at layer 3



(b) third stage: 1st attention head at layer 3



(c) first stage: 11th head at layer 12



(d) third stage: 11th head at layer 12

Figure 10.7: Comparison of attention heads between first and third stage prediction. Attention weights with larger absolute values have darker colors.

attribute's context.

Figure 10.7(d) illustrates the 11th attention head at layer 12 from the third stage prediction. This attention head clearly focuses on the token *from* in the context field to infer the schema label *to*. This attention head learns that the attribute *to* always occurs in a data table that contains the attribute *from*.

10.5 Summary

We have addressed the problem of semantic labeling using both data values and contextual information of an attribute. We have shown that a context-aware model that combines data values and column's context within the structured prediction framework outperforms methods that predict semantic labels only on the basis of data values. Our method SeLaB has been evaluated on three real-world datasets from multiple domains: WikiTables extracted from Wikipedia, Log tables from network equipment, and a combined dataset that contains general web tables. We have shown that the attribute's predicted context, which is incorporated into SeLaB after formulating the semantic labeling as a structured prediction problem, reduces the ambiguity in semantic labels prediction and outperforms the prior state of the art by 0.05 to 0.10 on most metrics, and by as much as 0.20 Macro-F on a dataset consisting primarily of numeric tables. Our model is trained end-to-end for both feature extraction and label prediction which reduces the human effort in semantic labeling.

In addition, we evaluated SeLaB on the masked headers task where we randomly mask a percentage of headers from data tables, and we predict the schema labels of the masked headers. SeLaB generates accurate predictions for the masked headers using both data values and the masked header's context. Automatic table extraction often leads to a percentage of missing or false headers which lowers the quality of the data table collections. SeLaB can be used as a data cleaning tool to predict missing headers in the extracted data tables, and improve the quality of the table collections for downstream tasks.

To explain the predictions of SeLaB, we conduct multiple experiments by comparing SeLaB embeddings and attention heads from multiple layers in both first and third stage prediction of the testing phase. Our analysis using the cosine similarity between the embedding of [CLS] token and the average of data values and context tokens shows that SeLaB considers both fields when inferring the third stage schema labels. Additionally, we compared multiple attention heads to show how SeLaB updates its prediction in the third stage using the attribute's context. Although BERT contains millions of parameters, our analysis using the attention heads helps to explain multiple predictions obtained from SeLaB.

Future work includes looking at how to incorporate metadata of each data table, such as table caption and description, into SeLaB, and how to select the best subset of data values for each column to improve semantic labeling results. Additionally, SeLaB can be pretrained on large collections of data tables, and explored in other table-related downstream tasks such as the table retrieval. SeLaB provides embeddings for the attributes of data tables which can be used to compute the retrieval score in table search.

Chapter 11

Domain Adaptation for Matching Entities

11.1 Introduction

Entity matching (EM) identifies data records that refer to the same real-world entity. EM is an important step in data cleaning and integration [40, 207], knowledge base enrichment [169], and entity linking [212]. Researchers have studied EM for many years in the context of data mining and integration.

In the past few years, deep learning (DL) has led to a significant improvement in multiple tasks, where DL-based methods achieved state-of-the-art (SOTA) results for text, image, and speech data. In many cases, DL models are trained end-to-end to automatically extract features and build predictive models. This significantly reduces the human effort that is needed in traditional methods for feature engineering, and gives the model the ability to capture specific features that are better than the hand-crafted ones for multiple tasks. Following the success of DL models, researchers have focused on exploring DL in data cleaning and integration. In particular, multiple DL methods have been proposed to solve the EM task[62, 70, 114, 165, 297]. Deep contextualized language models (DCLM), like BERT [58], RoBERTa [148], and DistilBERT [206] have been recently proposed to solve multiple tasks [253, 205, 36, 235, 239]. Building on DCLM, Ditto [139] achieved SOTA results in EM.

Although DL methods have led to a significant improvement in the EM task, these models need a huge amount of labeled data for each domain. DL-based models are trained in a supervised setting for each dataset in EM, where a different model is obtained and is fully fine-tuned on a specific dataset. This means that existing models capture the specific signals for each dataset in EM which leads to overfitting on just one dataset. In addition, the knowledge that is learned from one dataset is not explored to better understand the EM task so that the predictions in other datasets can be made with fewer labeled samples.

In order to overcome the limitations of prior methods, we propose a new method, called ***D****omain* ***A****daptation for* ***M****atching* ***E****ntities* (DAME), that transfers the task knowledge from multiple source domains to a target domain. Our method presents a new setting for EM where the objective is to capture task-specific knowledge from pretraining our model in multiple source domains, then testing our model on a target domain. In our study, we are interested in two aspects of our model. First, we study the zero-shot learning (ZSL) case of DAME on the target domain. Second, we study the effect of fine-tuning our proposed model on the target domain using different percentages of training data, and we compare our fine-tuned model to SOTA methods. We formulate EM as a mixture of experts with a global shared model [81, 119, 261] where each expert is trained on an individual source domain, and the global model is trained on all domains. Then, we aggregate the features from the experts using a global model-guided attention mechanism. We train DAME with unsupervised domain adaptation (DA) loss functions [81, 261] to reduce the domain shift between the source and target domains.

In summary, we make the following contributions: (1) We propose a new DA-based method for EM. Our new formulation of EM is based on the mixture of experts where we transfer learning from multiple source domains to a target domain. (2) We study the ZSL case on the target domain and demonstrate that our method learns the EM task and transfers the task knowledge to the target domain. (3) We extensively study fine-tuning our model on the target dataset from multiple domains, and demonstrate that our model generalizes better than SOTA methods for most of the datasets.

11.2 Problem statement

Our formulation of DA in EM task is based on the unsupervised multi-source DA setting which consists of K labeled source domains $\{\mathcal{S}_i\}_{i=1}^{K}$, where $\mathcal{S}_i = \left\{\left(x_j^{\mathcal{S}_i}, y_j^{\mathcal{S}_i}\right)\right\}_{j=1}^{|\mathcal{S}_i|}$ ($x_j^{\mathcal{S}_i}$ is the j-th instance of $\mathcal{S}_i$ with a label $y_j^{\mathcal{S}_i}$), and unlabeled target domain $\mathcal{T} =$

$\{x_j^{\mathcal{T}}\}_{j=1}^{|\mathcal{T}|}$. The objective is to learn a classifier M using labeled data from source domains and unlabeled data from the target domain so that (1) M produces accurate predictions on the target domain without fine-tuning (ZSL case), and (2) M generalizes better than SOTA methods on the target domain after partially or fully fine-tuning.

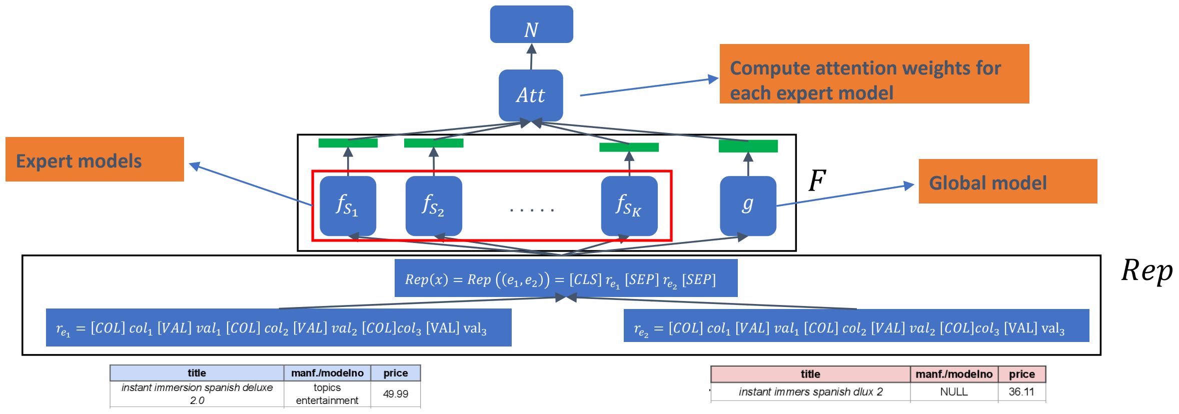
11.3 Domain adaptation for matching entities

In this section, we introduce our proposed method DAME which is a DA-based method for matching entities. We first describe the architecture of DAME, and then present the DA-based training strategy to update the parameters of our proposed model. Finally, we present our fine-tuning strategy in the case of using samples from the target domain to update DAME.

11.3.1 DAME architecture

There are multiple datasets that are available for the EM task. Therefore, our model is based on formulating the EM as a mixture of domain experts in the case of DA. Each expert model is trained on one source domain. We denote by f_{S_i}, the expert model that is trained on S_i. Training a mixture of experts and shared models improves the performance when multiple source domains are available as shown in prior works [81, 119, 261]. Therefore, we also add a global model g that is trained using all the source domains $\{\mathcal{S}_i\}_{i=1}^{K}$.

DCLM have been proposed in the DA setting to solve multiple tasks [261, 83, 86, 151, 200]. We propose to incorporate DCLM in our DA-based model to solve the EM task. Each f_{S_i} and g are initialized using DistillBERT [206]. We choose to use DistilBERT as the main component for the expert and global models for two reasons. First, by incorporating DCLM, we compare records in their entirety which has been shown to be more effective than attribute-based comparisons. Second, DistilBERT has a reduced size and comparable performance to BERT, and our objective is to include many source domains while keeping the time and memory complexity reasonable. The architecture of DAME is shown in Figure 11.1. In general, our proposed



N
Att
Compute attention weights for each expert model
Expert models
f_{S_1}
f_{S_2}
.....
f_{S_K}
g
F
Global model
Rep
$Rep(x) = Rep((e_1, e_2)) = [CLS]\ r_{e_1}\ [SEP]\ r_{e_2}\ [SEP]$
$r_{e_1} = [COL]\ col_1\ [VAL]\ val_1\ [COL]\ col_2\ [VAL]\ val_2\ [COL] col_3\ [VAL]\ val_3$
$r_{e_2} = [COL]\ col_1\ [VAL]\ val_1\ [COL]\ col_2\ [VAL]\ val_2\ [COL] col_3\ [VAL]\ val_3$

title	manf./modelno	price
instant immersion spanish deluxe 2.0	topics entertainment	49.99

title	manf./modelno	price
instant immers spanish dlux 2	NULL	36.11

Figure 11.1: Architecture of DAME for EM task. The *Rep* module takes a pair of records as inputs, and produces a BERT-based representation. The expert models $\{f_{S_i}\}_{i=1}^{K}$ are shown in the red box. The expert and global models within the feature extractor block F produce multiple embeddings, shown in green color, using the representation from *Rep* block as input. These embeddings are used in the attention block *Att* to produce a single feature vector. A classification layer N maps the output of *Att* to a matching score.

model M is composed of four modules:

$$M = N \circ Att \circ F \circ Rep \tag{11.1}$$

Rep is a representation module that produces the sequence input from a pair of records x, F is a feature extractor that produces multiple embeddings for the sequence input of the record pair x using expert models $\{f_{S_i}\}_{i=1}^{K}$ and the global model g, *Att* is an attention module that aggregates the embeddings of the expert models to produce the final multi-source embedding, and N is a classification layer that maps the final embedding to a confidence score to make a matching/non-matching decision on a record pair.

Representation module *Rep*

Each record pair $x = (e_1, e_2)$ is composed of two data entries $e_1 \in D_1$ and $e_2 \in D_2$ that correspond to candidate rows from two collections of data entries D_1 and D_2. **Both D_1 and D_2 are from the same source domain**. Each data entry $e_i = \{(\mathrm{attr}_j, \mathrm{val}_j)\}_{1 \le j \le C}$ is a set of attribute-value pairs denoted by $(\mathrm{att}_j, \mathrm{val}_j)$, where C

is the number of attributes in each record. We follow the encoding of Ditto [139] for serializing data entries to produce a sequence for each record from the attribute-value pairs:

$$r_{e_i} = \text{[COL] attr }_1\text{[VAL]val}_1 \dots \text{[COL] attr }_k\text{[VAL]val}_k \tag{11.2}$$

where [COL] and [VAL] are special tokens that denote the start of attributes and values, respectively. The input of EM is a pair of records $x = (e_1, e_2)$. So, *Rep* takes as input a pair of records, and produces a sequence pair of serialized entries that is given by:

$$Rep(x) = Rep((e_1, e_2)) = \text{[CLS]}r_{e_1}\text{[SEP]}r_{e_2}\text{[SEP]}, \tag{11.3}$$

where [SEP] and [CLS] are BERT special tokens that are added into the sequence similar to the sentence pair classification setting.

Feature extractor F

We have $K+1$ DistilBERT models: K expert models $\{f_{S_i}\}_{i=1}^{K}$ and a global shared model g. We use $Rep(x)$ as input to the $K+1$ models to extract K source domain-based embeddings denoted by $f_{S_i}(Rep(x)), i = 1, \dots, K$, and a global model-based embedding denoted by $g(Rep(x))$. The embeddings from the source domain models and the global model are extracted using the hidden state of the [CLS] token from the last Transformer block in each DistilBERT model. In conclusion, the output of F is given by:

$$F(Rep(x)) = \{f_{S_i}(Rep(x))\}_{i=1}^{K} \cup g(Rep(x)) \tag{11.4}$$

Attention module *Att*

When aggregating the embeddings that are extracted using F, the embeddings from the source domains and the global model should not be treated equally as there are domains that are more relevant to a given record pair x than others. We use a parameterized attention model that attends to all domains using a dot product-based attention where three parametric matrices are introduced: a query matrix $Q \in \mathbb{R}^{d\times d}$, a key matrix $K_e \in \mathbb{R}^{d\times d}$, and a value matrix $V \in \mathbb{R}^{d\times d}$, where d is the dimension of the embedding. We first concatenate all the embeddings from $F(Rep(x))$ to form an embedding matrix denoted by $E \in \mathbb{R}^{(K+1)\times d}$. The attention operations are defined by:

$$\begin{aligned}
&\alpha = g(Rep(x))^T Q \in \mathbb{R}^{1\times d}\\
&\mathcal{K} = EK_e \in \mathbb{R}^{(K+1)\times d}\\
&\mathcal{V} = EV \in \mathbb{R}^{(K+1)\times d}\\
&Att(Rep(x), Q, K, V) = \text{softmax}\left(\frac{\alpha\mathcal{K}^T}{\sqrt{d}}\right)\mathcal{V} \in \mathbb{R}^{1\times d}
\end{aligned} \tag{11.5}$$

An important design choice in our attention module *Att* is the use of the global representation $g(Rep(x))$ to compute the weights in the query matrix. Given that the global model is trained on all the source domains, we expect the global model's embedding to transfer to the target domain, and by consequence we obtain more accurate attention weights in the target domain to aggregate the source domains, mainly in the ZSL case. The output of the attention module is used as input to the classification layer N to predict the matching score of the input record pair x.

11.3.2 Training strategy

In the multi-source DA setting, we have K labeled source domains $\{\mathcal{S}_i\}_{i=1}^K$, where $\mathcal{S}_i = \left\{\left(x_j^{\mathcal{S}_i}, y_j^{\mathcal{S}_i}\right)\right\}_{j=1}^{|\mathcal{S}_i|}$, and an unlabeled target domain $\mathcal{T} = \left\{x_j^{\mathcal{T}}\right\}_{j=1}^{|\mathcal{T}|}$. Our training phase is based on the multi-task learning setting. In each batch for the training phase, we sample B pairs of records $X_j = (x_1^{\mathcal{S}_j}, y_1^{\mathcal{S}_j}), (x_2^{\mathcal{S}_j}, y_2^{\mathcal{S}_j}), \ldots, (x_B^{\mathcal{S}_j}, y_B^{\mathcal{S}_j})$ from a given source S_j. Our loss function $\mathcal{L}$ is composed of four parts and is given by:

$$\mathcal{L}(X_j) = \lambda_1\mathcal{L}_1(X_j) + \lambda_2\mathcal{L}_2(X_j) + \lambda_3\mathcal{L}_3(X_j) + \lambda_4\mathcal{L}_4(X_j) \tag{11.6}$$

where λ_1, λ_2, λ_3, and λ_4 are hyperparameters that control the contribution of each loss to the final loss function $\mathcal{L}$; each of $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$, and $\mathcal{L}_4$ represents a task-specific loss.

Expert domain loss $\mathcal{L}_1$

f_{S_i} represents the expert model for S_i for all $i \in 1, 2, \ldots, K$. To optimize each expert model f_{S_i}, we add a classification layer N_{S_i} that predicts the probabilities of matches and non-matches for each domain S_i. So in total we add K classification layers. Given that X_j is sampled from the j-th domain, the domain expert loss $\mathcal{L}_1$ is given by:

$$\mathcal{L}_1(X_j) = \frac{1}{B}\sum_{l=1}^{B} CrossEnt(N_{S_j}(f_{S_j}(Rep(x_l^{\mathcal{S}_j}))), y_l^{\mathcal{S}_j}) \tag{11.7}$$

where $CrossEnt$ denotes the cross entropy loss function.

Global model loss $\mathcal{L}_2$

The global model is trained on all the source domains in order to learn a universal embedding for the EM task that supports transfer to the target domain while maintaining important matching signals for each source domain. In addition, the embedding of the global model is multiplied with the query matrix Q in the attention module Att to compute the contribution of each source domain to the final representation. After learning how to aggregate features in the training phase on source domains, the global model guides the attention module Att to pick the most important source domains for the target domain during the testing phase. To optimize the global model g, we add a classification layer N_g that predicts the probabilities of matches and non-matches for all source domains. The global model loss $\mathcal{L}_2$ is given by:

$$\mathcal{L}_2(X_j) = \frac{1}{B}\sum_{l=1}^{B} CrossEnt(N_g(f_g(Rep(x_l^{\mathcal{S}_j}))), y_l^{\mathcal{S}_j}) \tag{11.8}$$

Meta-target loss $\mathcal{L}_3$

In DA, the objective is to incorporate multiple source domains to predict labels for samples from the target domain during the testing phase. In order to simulate the process of DA during the training phase, we use the meta-target and meta-sources similar to Guo et al. [81]. Given that X_j is sampled from the j-th domain, the meta-target is the j-th source domain and the meta-sources are $\{\mathcal{S}_i\}_{i=1,i\neq j}^{K}$. The meta-model M_{S_j} differs only on the feature extractor part F_{S_j} comparing to M which is given by:

$$M_{S_j} = N \circ Att \circ F_{S_j} \circ Rep \tag{11.9}$$

where:

$$F_{S_j}(Rep(x)) = \{f_{S_i}(Rep(x))\}_{i=1,i\neq j}^{K} \cup g(Rep(x)) \tag{11.10}$$

The same attention module *Att* is applicable to the output of the meta-feature extractor F_{S_j} where the query matrix based on the global model attends to all the expert embeddings in the key matrix regardless of the number of expert models. Finally, the meta-target loss $\mathcal{L}_3$ for the batch X_j is given by:

$$\mathcal{L}_3(X_j) = \frac{1}{B}\sum_{l=1}^{B} CrossEnt(M_{S_j}(x_l^{\mathcal{S}_j}), y_l^{\mathcal{S}_j}) \tag{11.11}$$

Adversarial loss $\mathcal{L}_4$

The global model g plays an important role in the attention module *Att*. Learning a domain invariant embedding from the global model makes the transfer to the target domain smoother as the attention weights should be more accurate. To obtain a domain invariant representation from g, we adapt the domain adversarial training for EM. Similar to the generative adversarial network (GAN), a min-max objective function is introduced to optimize the parameters of the generator which is the global model g and the discriminator denoted by D. The parameters of D are optimized to predict the domain of a sample x using $g(Rep(x))$, and the parameters of g are optimized to produce a confusing representation $g(Rep(x))$ for D. We alternate between updating D and g. Given that X_j is sampled from the j-th domain, in order to update D, we minimize $\mathcal{L}_D$ which is given by:

$$\mathcal{L}_D(X_j) = \frac{1}{B}\sum_{l=1}^{B} CrossEnt(D(f_g(Rep(x_l^{\mathcal{S}_j}))), j) \tag{11.12}$$

$\mathcal{L}_D$ is minimized with respect to only the parameters of D. Then, we set $\mathcal{L}_4(X_j) = -\mathcal{L}_D(X_j)$ to update the parameters of g when minimizing $\mathcal{L}$ (D parameters are fixed). Unlabeled samples $\mathcal{T} = \left\{x_j^{\mathcal{T}}\right\}_{j=1}^{|\mathcal{T}|}$ from the target domain can be also considered as an additional domain when updating the parameters of D and g by alternating between minimizing $\mathcal{L}_D$ and $-\mathcal{L}_D$, respectively. In this case, the total number of labels that are used in $\mathcal{L}_D$ is equal to $K + 1$.

11.3.3 Fine-tuning DAME on the target domain

During fine-tuning DAME on the target domain, we only update the weights of the global model g, attention weights *Att*, and the classification layer N, and we keep the weights of the expert models $f_{S_1}, f_{S_2}, \ldots, f_{S_K}$ frozen. The objective of the fine-tuning step is to slightly update the parameters of DAME to incorporate dataset-specific signals related to the target domain without changing the parameters of expert models. There are multiple fine-tuning scenarios on the target domain. First, we can use all the samples from the target domain or only a limited budget of samples for fine-tuning. Second, in the case of having access to only a limited budget of samples, we can randomly choose samples, or adapt active learning (AL) selection strategies to select the most promising samples. We experiment with all the scenarios and produce AL results using methods from [255, 72, 210].

11.4 Evaluation

11.4.1 Baselines

We compare the performance of our proposed model against the best performing method in the category of attribute-level comparators which is DeepMatcher [165] (the previous SOTA), and the SOTA in EM which is Ditto [139]. We are interested in two aspects of our proposed model DAME. First, we evaluate the ZSL case for DAME by comparing the results to the baselines that are fine-tuned on different percentages of training data. Second, we compare the results of fine-tuning DAME on the target domain against fine-tuning the baselines on the target domain.

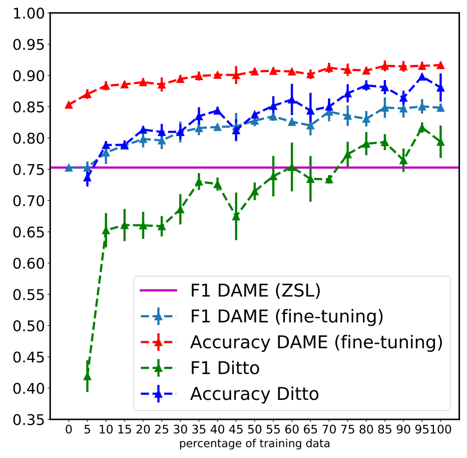
11.4.2 Experimental Setup

We evaluate the performance of DAME and baselines on the EM task using precision, recall, F1-score, and accuracy of predictions on the testing set. We write †, and ‡ to denote that the absolute difference between Ditto trained on 50% of data and DAME (ZSL) is less than 0.15, and less than 0.1, respectively. We write § to denote that either the absolute difference between Ditto trained on 50% of data and DAME (ZSL) is less than 0.05 or DAME (ZSL) is better than Ditto trained on 50% of data. DAME is trained for 3 epochs on the source domains. We compare fine-tuning results for DAME and baselines after training for 10 epochs on the same percentage of training data from the target domain. The hyperparameters λ_1, λ_2, λ_3, and λ_4 are fine-tuned for one dataset and then kept the same for all the experiments. We distinguish 3 sets of experiments based on the structure of datasets. The first set of experiments studies DA for shoes, cameras, computers, and watches. These datasets have a unique attribute which is *title*. The second set of experiments also studies DA for datasets that have similar structures which are DBLP-GoogleScholar and DBLP-ACM. The set of attributes for these two datasets are *title*, *authors*, *venue*, and *year*. The third set of experiments is related to *DA in the wild* where we study DA using all 12 datasets regardless of the structures and domains.

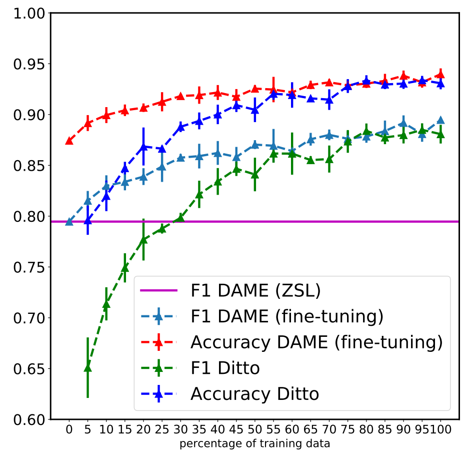
11.4.3 Experimental results

DA for shoes, computers, watches, cameras

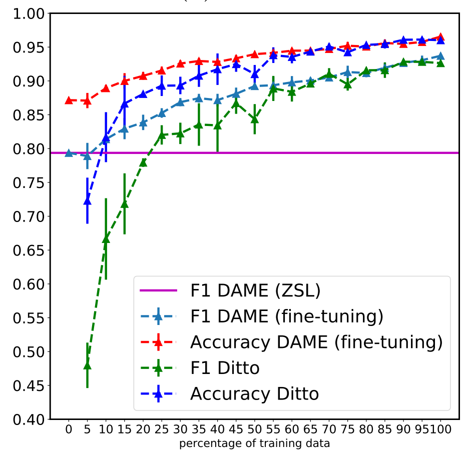
Figure 11.2 shows the comparison of DAME results against Ditto for shoes, computers, watches, and cameras. The caption of each subfigure represents the target domain, and the remaining 3 domains represent the source domains. Each data point represents the mean of 5 trials, and the vertical line in each data point represents the standard deviation (std). The plots report two evaluation metrics: F1 score and accuracy. In all figures, the light blue plot represents the F1 score of DAME, and is compared against the green plot that represents the F1 score of Ditto; the red plot represents the accuracy of DAME, and is compared against the blue plot that represents the accuracy of Ditto. DAME and Ditto outperform DeepMatcher for all evaluation metrics, so that we only include DAME and Ditto results to avoid clutter in the figures. The magenta color represents the F1 score of the DAME (ZSL) for the target domain, which is equivalent to 0% of supervised training data from the target domain. We achieve high F1 scores for DAME (ZSL) for both shoes and cameras datasets, where the F1 score for DAME (ZSL) is equivalent to fine-tuning Ditto on 72% and 85% of training data for the shoes and cameras, respectively. The results are lower for computers and watches where the F1 score of DAME (ZSL) is equivalent to Ditto fine-tuned on around 25% of training data. Figure 11.2 shows the results of fine-tuning DAME using different percentages of training data. Fine-tuning DAME leads to a better and more stable (smaller std in most fractions of the training data) performance than Ditto for all datasets which means that DAME generalizes better than existing methods in EM for datasets with similar structure. This can be explained by the important role of DA in learning the task so that the weights are better warmed up for EM.



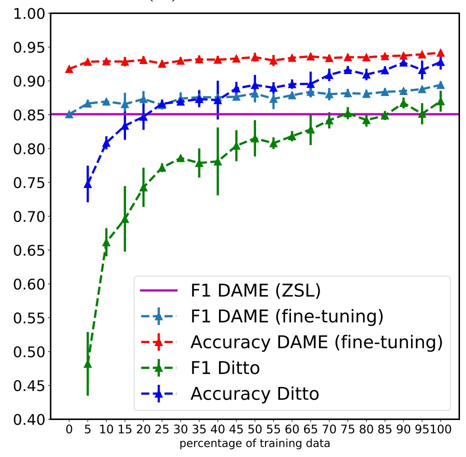
(a) Shoes



(b) Computers



(c) Watches



(d) Cameras

Figure 11.2: Comparison of DAME results against Ditto for datasets with similar structure (shoes, computers, watches, and cameras). The plots report two evaluation metrics: F1 score and accuracy. In all figures, the light blue plot represents the F1 score of DAME, and is compared against the green plot that represents the F1 score of Ditto; the red plot represents the accuracy of DAME, and is compared against the blue plot that represents the accuracy of Ditto; the magenta color represents the F1 score of the ZSL for the target domain.

DA for DBLP-GoogleScholar, DBLP-ACM

Table 11.1 summarizes the performance of different approaches on the second set of datasets with the same structure which is composed of DBLP-GoogleScholar and DBLP-ACM. In this case, we have one target dataset and one source dataset. We achieve high results for DAME (ZSL) for both datasets. In addition, fine-tuning DAME slightly increases the F1 and accuracy for both datasets. So, consistent with the first set of experiments, we conclude that DAME transfers the task knowledge

Method Name	Precision	Recall	F1	Accuracy
DeepMatcher [165]	**0.9489**	0.9373	0.9431	0.9789
Ditto [139]	0.9358	0.9542	0.9449	0.9793
DAME (ZSL)	0.9098	0.8579	0.8831	0.9576
DAME (full fine-tuning)	0.9354	**0.9719**	**0.9533**	**0.9850**

(a) DBLP-GoogleScholar

Method Name	Precision	Recall	F1	Accuracy
DeepMatcher [165]	0.9855	0.9869	0.9861	0.9945
Ditto [139]	**0.9865**	0.9865	0.9865	0.9951
DAME (ZSL)	0.8769	**0.9954**	0.9324	0.9741
DAME (full fine-tuning)	**0.9865**	**0.9954**	**0.9909**	**0.9971**

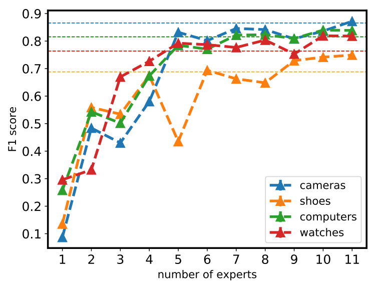
(b) DBLP-ACM

Table 11.1: DA results for EM using datasets with similar structure. (a) the target dataset is DBLP-GoogleScholar and the source dataset is DBLP-ACM; (b) the target dataset is DBLP-ACM and the source dataset is DBLP-GoogleScholar.

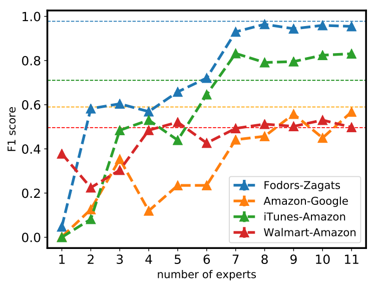
from the source domains to a target domain in the case of datasets with similar structures.

DA in the wild

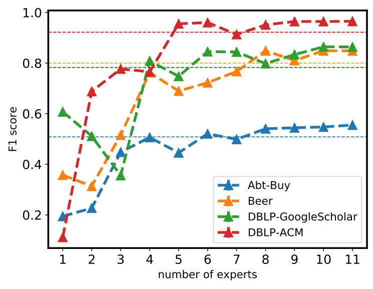
We study the case of transferring knowledge between datasets with different domains and structures. We call this setting DA in the wild which simulates real-world scenarios. Tables 11.3 and 11.4 (end of the chapter) show extensive experiments on 12 datasets reporting evaluation metrics for multiple methods. DAME (ZSL) achieves a better F1 score than DeepMatcher fine-tuned with 50% of training data from the target domain for 7 out of 12 datasets. The absolute difference between the F1 score of Ditto trained on 50% of data and DAME (ZSL) is less than 0.1, and 0.05 for 83% and 41% of datasets, respectively. By comparing the F1 score of fine-tuning all methods using 50% of training data from the target domain, we achieve SOTA results for 10 out of 12 datasets. By comparing the F1 score of fine-tuning all methods using all training data from the target domain, we achieve SOTA results for 10 out of 12 datasets. This means that DAME generalizes better than existing methods for



(a)



(b)



(c)

Figure 11.3: Comparison of F1 score results with different numbers of expert domains against using global model representation during testing phase on the target domain.

datasets in the wild.

Expert models vs Global model

Figure 11.3 shows the comparison of F1 score results with different numbers of expert domains against using the global model representation during the testing phase on the target domain in the case of ZSL. The x-axis represents the number of experts that we use for predictions. For example, if the number of experts is equal to 6, it means that we randomly choose 6 experts and we drop the remaining 5 experts. Each data point in Figure 11.3 represents an average of 5 trials. The dashed line represents the F1 score for the global model. For 10 out of 12 datasets, combining multiple experts using the attention network *Att* leads to better results than the global model. Figure 11.3 shows that the least and largest number of experts needed to outperform the global model equals 5 (DBLP-ACM) and 11 (cameras), respectively. Overall, we obtain better F1 scores for the mixture of experts when we increase the number of experts. This means that the experts help to better understand the EM task, and therefore transfer the learned task knowledge to the unseen target domain.

DAME with Active learning

So far, we have discussed the performance of fine-tuning DAME using randomly selected samples from the target domain. To improve the results of fine-tuning our model, we investigate multiple AL selection techniques given a limited budget of labeled instances. Table 11.2 shows the results of multiple AL selection methods applied to the DAME (ZSL) model. The starting point is our DA-based model which

Table 11.2: F1 results for AL after DA.

Method	Shoes	Computers	Watches	Cameras
DAME (ZSL)	0.7527	0.7946	0.7936	0.8507
DAME (full fine-tuning)	0.8483	0.8947	0.9371	0.8941
Random Sampling (5%)	0.7527	0.8181	0.8004	0.8664
Least Confidence [255] (5%)	0.7818	0.8402	0.8209	0.8745
Entropy Sampling [255] (5%)	0.7859	0.8464	0.8166	0.8748
USDE [72] (5%)	**0.7877**	0.8437	0.8151	**0.8775**
BALD [72] (5%)	0.7852	**0.8472**	**0.8313**	0.8705
K-Centers Greedy [210] (5%)	0.7674	0.8271	0.8206	0.8687
K-Means [210] (5%)	0.7527	0.8042	0.8097	0.8596
Core-Set [210] (5%)	0.7621	0.8304	0.8168	0.8734
Random Sampling (25%)	0.8120	0.8418	0.8528	0.8741
Least Confidence [255] (25%)	0.8228	0.8804	0.8677	0.8888
Entropy Sampling [255] (25%)	0.8207	0.8770	0.8740	0.8925
USDE (25%) [72]	**0.8286**	0.8741	0.8688	0.8842
BALD (25%) [72]	0.8247	**0.8835**	**0.8872**	**0.8941**
K-Centers Greedy [210] (25%)	0.8155	0.8771	0.8869	0.8780
K-Means [210] (25%)	0.8057	0.8658	0.8694	0.8737
Core-Set [210] (25%)	0.8161	0.8696	0.8812	0.8776

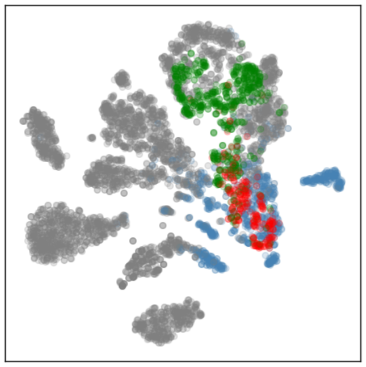
is not fine-tuned on the target domain, and the best performance corresponds to DAME fine-tuned on all training data from the target domain. We report results using two budget levels: 5% and 25% of the training data from the target domain. The simplest baseline is Random Sampling. The remaining baselines can be categorized into two groups: the confidence-based baselines which are: Least Confidence [255], Entropy Sampling [255], Uncertainty Sampling with Dropout Estimation (USDE) [72], and Bayesian Active Learning Disagreement (BALD) [72]; and the embedding-based baselines which are K-Centers Greedy [210], K-Means [210], and Core-Set [210]. The selection of samples in the first group is based on the confidence scores of the training data from the target domain that are computed using the DAME (ZSL) model. For example, for a budget of b samples, Least Confidence corresponds to the top b samples with the lowest confidence level. Multiple predictions for a given sample are needed for USDE and BALD to compute the uncertainty functions, and we obtain these different predictions by activating the dropout layers during the inference phase on the target domain. The second group is based on the embeddings of samples from the target domain that are obtained from the DAME (ZSL) model. Clustering of the input space is then applied to determine centers of clusters or core sets. Table 11.2 shows that the confidence-based methods lead to better results than the embedding-based methods. In particular, when we select 25% of samples using the BALD method for the cameras dataset, we achieve the same F1 score of a fully fine-tuned DAME model using all training data from the target domain. This indicates that the predictions from the classification layer N of our model M accurately reflect the data points where DA was unsuccessful. Therefore, by fine-tuning on these samples from the training data, our model generalizes better on the testing set of the target domain.

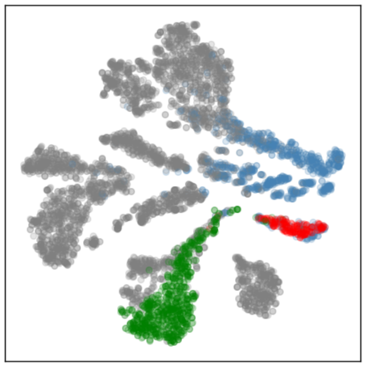
Visualization

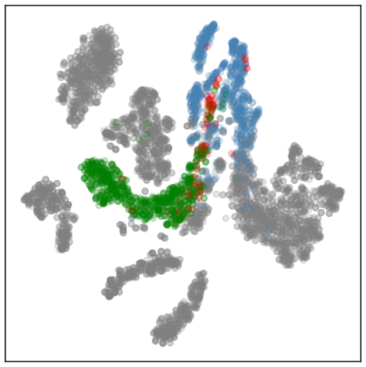
We finish our experiments by showing the embedding of DAME in the case of ZSL. Figure 11.4 shows the t-SNE visualization of the final embeddings for the target and source domains after DA in the wild (ZSL case). We only show the embeddings of six domains, but we notice similar patterns for all the datasets. The gray and blue colors represent randomly selected data points from the source domains with a label 0 and label 1, respectively; the green and red colors represent randomly selected data points from the testing set of the target domain with a label 0 and label 1, respectively. 12 domains are used in each experiment, where the caption of each subfigure represents the target domain, and the 11 remaining datasets represent the source domains. The best case is to have a mixture of blue and red dots which represent the matching class for the source and target domains, respectively, and a mixture of gray and green dots which represent the non-matching class for the source and target domains, respectively. This means that we transfer the task knowledge from sources to the target domain for both labels. For example, for computers and DBLP-ACM, we obtain embeddings that respect the matching and non-matching classes as shown in Figure 11.4 (a) and (b). On the other hand, for Amazon-Google and Walmart-Amazon, there are green dots that are closer to the blue dots as shown in Figure 11.4 (c) and (d), and this leads to incorrect predictions.

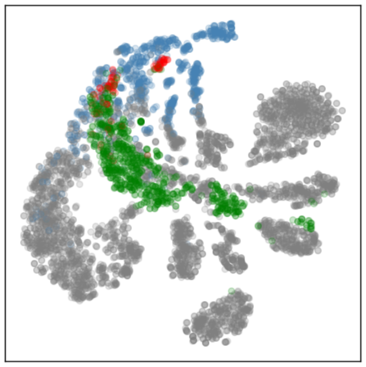
11.5 Summary

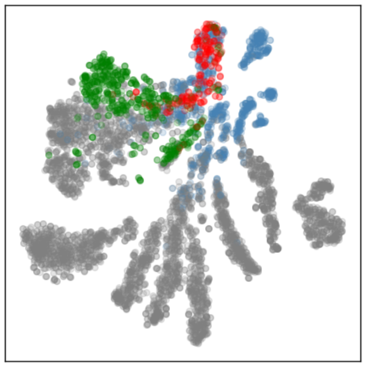
We have shown that our proposed model transfers learning from multiple source domains to an unseen target domain in the EM task. We formulate the EM task as a mixture of experts that capture task-specific knowledge from pretraining on multiple


(a) Computers


(b) DBLP-ACM


(c) Amazon-Google


(d) Walmart-Amazon


(e) Watches

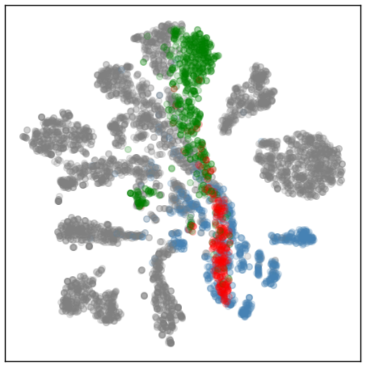

(f) Cameras

Figure 11.4: The t-SNE visualization of the final embeddings for the target and source domains after DA in the wild (ZSL case).

source domains and testing on a target domain. We evaluate DAME in two aspects. First, we study the ZSL case on the target domain and demonstrate that DAME learns the EM task and transfers knowledge to the target domain. Second, we study fine-tuning DAME on the target domain and demonstrate that DAME generalizes better than SOTA methods for most of the datasets. We showed that our results hold in two scenarios which are EM for datasets with similar structure and EM in the wild. Our experimental section contains extensive experiments over 12 datasets with different domains, sizes and structures. In addition, we showed the importance of selecting a specific set of samples in the fine-tuning of the target domain by studying AL methods with limited budget. Future work includes extending our model to pairs of records with different sets of attributes, and enriching our DA-based model with external knowledge, such as knowledge graphs, to better understand the EM task and therefore transfer more knowledge to the target domain.

Table 11.3: DAME results for EM in the wild. 12 domains are used in each experiment, where each row of the table represents the target domain, and the 11 remaining datasets represent the source domains. We write †, and ‡ to denote that the absolute difference between Ditto trained on 50% of data and DAME (ZSL) is less than 0.15, and less than 0.1, respectively. We write § to denote that either the absolute difference between Ditto trained on 50% of data and DAME is less than 0.05 or DAME is better.

Target dataset	Method	Precision	Recall	F1	Accuracy
Fodors-Zagats	DAME (ZSL)	0.9565§	1.0000§	0.9777§	0.9947§
	DeepMatcher[165] (50% training data)	0.9360	0.8333	0.8801	0.9735
	Ditto [139] (50% training data)	**1.0000**	0.9545	0.9767	**0.9947**
	DAME (50% training data)	0.9565	**1.0000**	**0.9777**	**0.9947**
	DeepMatcher[165] (full training data)	0.9092	0.9848	0.9437	0.9858
	Ditto [139] (full training data)	**1.0000**	0.9545	0.9767	0.9947
	DAME (full fine-tuning)	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Beer	DAME (ZSL)	0.7368§	1.000§	0.8484§	0.9450§
	DeepMatcher[165] (50% training data)	**0.8095**	0.4047	0.5396	0.8937
	Ditto [139] (50% training data)	0.7211	0.6428	0.6794	0.9065
	DAME (50% training data)	0.7801	**1.000**	**0.8758**	**0.9560**
	DeepMatcher[165] (full training data)	0.8183	0.7142	0.7588	0.9304
	Ditto [139] (full training data)	**0.8174**	0.9285	0.8660	**0.9560**
	DAME (full fine-tuning)	0.7801	**1.000**	**0.8758**	**0.9560**
iTunes-Amazon	DAME (ZSL)	0.6750	1.000§	0.8059‡	0.8807‡
	DeepMatcher[165] (50% training data)	0.9005	0.7901	0.8406	0.9266
	Ditto [139] (50% training data)	0.8685	0.8518	0.8594	0.9311
	DAME (50% training data)	**0.9333**	**0.9629**	**0.9467**	**0.9724**
	DeepMatcher[165] (full training data)	0.9139	0.9135	0.9135	0.9571
	Ditto [139] (full training data)	0.9282	0.9259	0.9258	0.9633
	DAME (full fine-tuning)	**0.9807**	**0.9259**	**0.9524**	**0.9770**
Abt-Buy	DAME (ZSL)	0.4545	0.6796†	0.5447	0.8778‡
	DeepMatcher[165] (50% training data)	0.6978	0.5355	0.6033	0.9244
	Ditto [139] (50% training data)	0.7916	0.7839	0.7870	0.9543
	DAME (50% training data)	**0.7960**	**0.7864**	**0.7911**	**0.9553**
	DeepMatcher[165] (full training data)	0.7382	0.6181	0.6725	0.9352
	Ditto [139] (full training data)	**0.9206**	0.7864	**0.8481**	**0.9697**
	DAME (full fine-tuning)	0.8243	**0.8592**	0.8410	0.9650
Amazon-Google	DAME (ZSL)	0.5431	0.6453§	0.5898‡	0.9084§
	DeepMatcher[165] (50% training data)	0.5623	0.5327	0.5416	0.9085
	Ditto [139] (50% training data)	**0.7055**	0.6709	**0.6877**	**0.9378**
	DAME (50% training data)	0.6339	**0.7435**	0.6809	0.9291
	DeepMatcher[165] (full training data)	0.7002	0.6011	0.6454	0.9326
	Ditto [139] (full training data)	0.6709	**0.8098**	0.7338	0.9400
	DAME (full fine-tuning)	**0.7046**	0.7692	**0.7353**	**0.9435**
Shoes	DAME (ZSL)	0.6798§	0.8135§	0.7407§	0.8450§
	DeepMatcher[165] (50% training data)	0.6346	0.7163	0.6719	0.8096
	Ditto [139] (50% training data)	0.7137	0.7559	0.7301	0.8496
	DAME (50% training data)	**0.8234**	**0.8423**	**0.8325**	**0.9077**
	DeepMatcher[165] (full training data)	0.6908	0.7988	0.7400	0.8468
	Ditto [139] (full training data)	0.7569	0.8389	0.7950	0.8819
	DAME (full fine-tuning)	**0.8421**	**0.8796**	**0.8600**	**0.9220**

Table 11.4: DAME results for EM in the wild (cont).

Target dataset	Method	Precision	Recall	F1	Accuracy
	DAME (ZSL)	$0.7957^{\S}$	$0.8729^{\S}$	$0.8325^{\S}$	$0.9043^{\S}$
	DeepMatcher[165] (50% training data)	0.5762	0.7547	0.6529	0.7820
	Ditto [139] (50% training data)	0.8020	**0.9080**	0.8517	0.9139
Computers	DAME (50% training data)	**0.8303**	0.9063	**0.8659**	**0.9234**
	DeepMatcher[165] (full training data)	0.7002	0.8350	0.7614	0.8576
	Ditto [139] (full training data)	**0.8682**	0.9147	0.8908	0.9389
	DAME (full fine-tuning)	0.8630	**0.9264**	**0.8935**	**0.9398**
	DAME (ZSL)	$0.7267^{\dagger}$	$0.9124^{\S}$	$0.8090^{\ddagger}$	$0.8834^{\ddagger}$
	DeepMatcher[165] (50% training data)	0.6997	0.7274	0.7126	0.8415
	Ditto [139] (50% training data)	0.8664	0.8996	0.8827	0.9352
Watches	DAME (50% training data)	**0.8691**	**0.9160**	**0.8917**	**0.9397**
	DeepMatcher[165] (full training data)	0.7771	0.8309	0.8030	0.8896
	Ditto [139] (full training data)	**0.9145**	0.9178	0.9161	0.9545
	DAME (full fine-tuning)	0.9010	**0.9470**	**0.9234**	**0.9575**
	DAME (ZSL)	$0.8376^{\S}$	$0.8958^{\S}$	$0.8657^{\S}$	$0.9243^{\S}$
	DeepMatcher[165] (50% training data)	0.5896	0.6863	0.6328	0.7842
	Ditto [139] (50% training data)	0.7585	0.8628	0.8020	0.8831
Cameras	DAME (50% training data)	**0.8801**	**0.8871**	**0.8825**	**0.9356**
	DeepMatcher[165] (full training data)	0.6986	0.7847	0.7388	0.8486
	Ditto [139] (full training data)	0.8573	0.9062	0.8809	0.9333
	DAME (full fine-tuning)	**0.8917**	**0.9070**	**0.8963**	**0.9432**
	DAME (ZSL)	0.3558	$0.9015^{\S}$	0.5102	$0.8369^{\dagger}$
	DeepMatcher[165] (50% training data)	0.6938	0.5474	0.6118	0.9346
	Ditto [139] (50% training data)	**0.8501**	0.7098	0.7721	0.9607
Walmart-Amazon	DAME (50% training data)	0.8082	**0.8083**	**0.8082**	**0.9638**
	DeepMatcher[165] (full training data)	0.6971	0.6010	0.6448	0.9376
	Ditto [139] (full training data)	**0.8883**	0.7694	**0.8227**	**0.9687**
	DAME (full fine-tuning)	0.8615	**0.7875**	0.8226	0.9680
	DAME (ZSL)	$0.9077^{\S}$	$0.8490^{\ddagger}$	$0.8737^{\ddagger}$	$0.9499^{\S}$
	DeepMatcher[165] (50% training data)	0.9347	0.9439	0.9385	0.9770
	Ditto [139] (50% training data)	0.9356	**0.9448**	0.9385	0.9771
DBLP-GoogleScholar	DAME (50% training data)	**0.9367**	0.9411	**0.9389**	**0.9771**
	DeepMatcher[165] (full training data)	**0.9489**	0.9373	0.9431	0.9789
	Ditto [139] (full training data)	0.9358	**0.9542**	0.9449	0.9793
	DAME (full fine-tuning)	0.9392	0.9537	**0.9464**	**0.9798**
	DAME (ZSL)	$0.8661^{\dagger}$	$0.9854^{\S}$	$0.9219^{\ddagger}$	$0.9651^{\S}$
	DeepMatcher[165] (50% training data)	0.9787	0.9763	0.9774	0.9919
	Ditto [139] (50% training data)	**0.9865**	**0.9865**	**0.9865**	**0.9951**
DBLP-ACM	DAME (50% training data)	0.9787	0.9831	0.9809	0.9931
	DeepMatcher[165] (full training data)	0.9855	0.9869	0.9861	0.9945
	Ditto [139] (full training data)	**0.9865**	0.9865	0.9865	0.9951
	DAME (full fine-tuning)	0.9865	**0.9868**	**0.9866**	**0.9951**

Chapter 12

Conclusions

12.1 Summary

In this dissertation, we first presented the literature work that is related to our proposed methods. We discussed both the related deep learning techniques and the problems that we have addressed in dataset search and curation. In terms of neural components, we introduced the convolutional neural network, recurrent neural network, long short-term memory, gated recurrent units, and attention mechanism. We also summarized the embedding techniques including the traditional, contextualized, and deep contextualized embeddings. In addition, we introduced a recent line of research that focuses on embeddings obtained from graph representations. Our work is related to search engines in general, so we introduced the neural ranking models that are used in document retrieval. In terms of machine learning concepts, we introduced the N-gram language models that are used to compute ranking scores. In addition, we presented the structured prediction task that is adapted in our new setting for semantic labeling. Finally, we presented the domain adaptation which is used to solve the entity matching task.

Then, we discussed the literature of table search, table similarity, semantic labeling, and entity matching, and we introduced the datasets that are used in our experiments to compare our proposed methods against baselines for table search, table similarity, semantic labeling, and entity matching.

After introducing the datasets, we presented our contributions in dataset search and curation. Extracting useful knowledge from datasets in the wild is a cumbersome process, and data scientists spend the majority of their time searching and cleaning datasets for downstream analytic tasks. Retrieving datasets, that are relevant to the user's information need, is the first step in the pipeline of data management in general. We discussed our proposed methods for dataset search, and addressed multiple research questions that we have raised. We summarize the main contributions in dataset search as follows:

- In Chapter 5, we propose an unsupervised method for table ranking, called MCON [241], where we build a new model for word embeddings of the tokens of table attributes using contextual information of every table. We demonstrate the usefulness of an attribute's collection of values (the data tokens) in creating a meaningful semantic representation of the attribute. We predict the context of tables using the trained contextual model, and we use a mixed ranking model that incorporates the metadata of a table and the additional contexts in order to calculate the retrieval score. We achieved 6% improvement in NDCG@5 over the best unsupervised baseline.

- In Chapter 6, we propose a new knowledge graph, called MultiEM-RGCN [238], that incorporates both dataset-dependent and dataset-agnostic knowledge from table corpus to incorporate multiple matching signals and external resources and learn embeddings for large collections of data tables. External semantic and lexical resources are used for edges and nodes leading to an heterogeneous graph. Multiple types of embeddings are learned simultaneously from our proposed knowledge graph using graph neural networks (GNN) with the link prediction pre-training task. Our proposed graph provides multiple embeddings for each token in all the fields. The new graph embeddings are incorporated into a learning to rank (LTR) architecture that combines multiple embeddings from our heterogeneous graph to solve the table retrieval task. We achieved 4% improvement in NDCG@5 over the best embedding baseline.

- In Chapter 7, we propose a deep semantic and relevance matching model, called DSRMM [237], that is able to capture multiple levels of semantic signals between query and table. We demonstrate the usefulness of query relevance-specific components for the table retrieval task. Using kernel pooling, we learn a feature vector based on the probability distribution of the similarity of each document token to each query token, and we learn the contribution of each token to the final relevance score using a Term Gating Network. Each of these components lead to improvement on retrieval tasks without leading to a large increase in the number of parameters of the model. DSRMM outperforms the best previously published results in table retrieval using STR [293], achieving up to 10% improvement in NDCG@5 score.

- In Chapter 8, we propose a new structure-aware BERT model, called StruBERT [240], that fuses the structural and textual information of a data table to produce four context-aware features: two fine-grained structure and context aware representations for rows and columns, and two coarse-grained representations for row and column guided [CLS] embedding. We propose a new ranking model, called miniBERT, that operates directly on the embedding level sequences formed from StruBERT features to solve three table-related downstream tasks which are: keyword- and content-based table retrieval, and table similarity. In addition to achieving better retrieval and classification results than existing baselines, the main benefit of StruBERT is solving both scenarios for table search: keyword- and content-based table retrieval.

After extracting relevant datasets, the next step is to automate the data curation in order to improve the quality of the data for downstream tasks. In particular, we focused on the semantic labeling, table similarity, and entity matching, and we presented multiple deep learning based approaches to solve these tasks. We summarize the main contributions in data curation as follows:

- In Chapter 10, we propose a new context-aware semantic labeling approach, called SeLaB [236], that takes into account data values and contextual information of attributes to obtain a context-sensitive embedding for semantic labeling. Our new formulation of semantic labeling is based on the structured prediction setting in which the input to our model is a data table with missing headers, and we sequentially generate schema labels for each data table. We incorporate data values and predicted contexts using BERT, which is trained end-to-end for feature extraction and label prediction. This reduces human effort in semantic labeling. To explain the predictions of SeLaB in semantic labeling, we compared SeLaB embeddings and attention heads from multiple layers in both first and third stage prediction of the testing phase. To show that SeLaB considers both data values and contextual information when inferring the third stage schema labels, we included the analysis about the cosine similarity between the embedding of the [CLS] token and the average of data values and context tokens. SeLaB improves upon the state-of-the-art approach Sherlock [101] for semantic labeling achieving 10% improvement in top-1 accuracy over Sherlock for WikiTables, and 20% improvement in top-1 accuracy over Sherlock for Log tables.

- In Chapter 8, we propose predicting the semantic matching between tables based on StruBERT [240] features with miniBERT ranking model as in the content-based table retrieval case.
- In Chapter 11, we propose a new domain adaptation-based method for entity matching called DAME [242]. Our new formulation of entity matching is based on the mixture of experts where we transfer learning from multiple source domains to a target domain. We study the zero-shot learning case on the target domain and demonstrate that our method learns the entity matching task and transfers the task knowledge to the target domain. We extensively study fine-tuning our model on the target dataset from multiple domains, and demonstrate that our model generalizes better than state-of-the-art methods for most of the datasets. To reduce the number of fine-tuning samples in the target domain, DAME is fine-tuned on a limited budget of data by incorporating active learning techniques to select the best subset of samples for fine-tuning. In addition to achieving better classification results than existing baselines, the main benefit of this work is showing that the domain adaptation in the wild is a promising research direction.

12.2 Future Directions

The methods that are presented in this dissertation can be used to build a dataset search and curation system. The workflow for this system starts from a given user's query, that can be either keywords- or content-based query. Then, the search engine part of the system finds relevant datasets, and the curation part of the system returns a curated set of relevant datasets to the user. As discussed in Chapter 9, a multi-stage ranking architecture for dataset search is suitable for the trade-off between retrieval results and computation time in the deployment phase. The first stage consists of extracting the candidate datasets using simple combined ranking scores from MCON [241] and MultiEm-RGCN [238]. In this stage, recall is more important than precision to cover all possible relevant datasets. The irrelevant datasets can be discarded in the next stages. Then, for the second stage, the top ranked datasets from the first stage are re-ranked using DSRMM [237] to obtain a better set of relevant datasets. DSRMM is a supervised ranking method composed of convolutional filters and kernel pooling, and these operations are computationally efficient both in terms of time and memory. Finally, in the third stage, StruBERT [240] is used to rank the top datasets from the second phase, and return the final top ranked dataset to the user. This multi-stage model can reduce the number of datasets that should be ranked with StruBERT, which is computationally expensive in terms of time and memory. The curation part of the system applies data curation techniques to return a set of curated datasets. For instance, our system can predict more consistent header names for datasets using SeLaB [236], and find redundant records in datasets or similar records across datasets using DAME [242], in order to facilitate the data integration.

We can improve the dataset search and curation system by incorporating three components. First, including visualization tools helps the user to visualize multiple statistics about the datasets. Second, the system can be improved to retrieve answers for questions from users instead of the keyword-based queries, and this related to the question-answering over datasets [92, 30, 91]. Third, the system can be used to help researchers to find relevant scientific datasets to their work. This is a challenging task because there are no conventions or standards about research datasets. As an initial solution, we have proposed to build an ontology, called Machine Learning Progress Ontology (MLPO) [35], to track tasks, datasets, and metrics in machine learning. This ontology can be automatically populated by finding the relevant tasks, datasets, and metrics in research papers. So, our final system combines techniques from both information retrieval and semantic web in order to automate extracting useful knowledge from datasets. Such a system can reduce human effort by automating searching and cleaning datasets in order to improve the quality of data for data-driven downstream tasks.

Bibliography

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*, page 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] Hiteshwar Kumar Azad and Akshay Deepak. Query expansion techniques for information retrieval: A survey. *Inf. Process. Manag.*, 56(5):1698–1735, 2019.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[4] Nils Barlaug and Jon Atle Gulla. Neural networks for entity matching: A survey. *ACM Trans. Knowl. Discov. Data*, 15(3):52:1–52:37, 2021.

[5] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, editors, *Advances in Neural Information Processing Systems 19*, pages 137–144. MIT Press, 2006.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[7] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML*, volume 382, pages 41–48. ACM, 2009.

[8] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Rec.*, 28(1):54–59, 1999.

[9] Chandra Bhagavatula, Thanapon Noraset, and Doug Downey. Tabel: Entity linking in web tables. In *International Semantic Web Conference*, pages 425–441, 2015.

[10] 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. ACM, 2013.

[11] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48. ACM, 2003.

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics*, 5:135–146, 2017.

[13] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, page 1247–1250. Association for Computing Machinery, 2008.

[14] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[15] Jane Bromley, James Bentz, Leon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Sackinger, and Rookpak Shah. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:25, 08 1993.

[16] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 89–96. ACM, 2005.

[17] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proc. VLDB Endow.*, 2(1):1090–1101, August 2009.

[18] 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.

[19] 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.

[20] Matteo Cannaviccio, Denilson Barbosa, and Paolo Merialdo. Towards annotating relational data on the web with language models. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1307–1316. ACM, 2018.

[21] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 129–136. ACM, 2007.

[22] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012, 2018.

[23] R. Castro Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 989–1000, 2018.

[24] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. Structured learning for non-smooth ranking losses. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 88–96. ACM, 2008.

[25] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 4960–4964. IEEE, 2016.

[26] Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13:216–235, 2009.

[27] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the CNN/daily mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2358–2367, Berlin, Germany, August 2016. Association for Computational Linguistics.

[28] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles A. Sutton. Learning semantic annotations for tabular data. In *IJCAI*, 2019.

[29] Jiaoyan Chen, Ernesto Jimenez--Ruiz, Ian Horrocks, and Charles Sutton. Colnet: Embedding the semantics of web tables for column type prediction. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[30] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. Open question answering over tables and text. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[31] Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume

EMNLP 2020 of *Findings of ACL*, pages 1026–1036. Association for Computational Linguistics, 2020.

[32] Zhiyu Chen, Harini Eavani, Wenhu Chen, Yinyin Liu, and William Yang Wang. Few-shot NLG with pre-trained language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 183–190. Association for Computational Linguistics, 2020.

[33] 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*, page 1515–1522. International World Wide Web Conferences Steering Committee, 2018.

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

[35] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, and Brian 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 19th International Semantic Web Conference (ISWC 2020), Globally online, November 1-6, 2020 (UTC)*, volume 2721 of *CEUR Workshop Proceedings*, pages 232–237. CEUR-WS.org, 2020.

[36] 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*, page 589–598, New York, NY, USA, 2020. Association for Computing Machinery.

[37] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Dawei Yin, and Brian D. Davison. MGNETS: multi-graph neural networks for table search. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 2945–2949. ACM, 2021.

[38] Zhiyu Chen, Shuo Zhang, and Brian D. Davison. WTR: A test collection for web table retrieval. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 2514–2520. ACM, 2021.

[39] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc, of the Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, October 2014.

[40] Peter Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1065–1068. ACM, 2008.

[41] Peter Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Data-Centric Systems and Applications. Springer, 2012.

[42] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.

[43] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[44] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[45] Kevyn Collins-thompson, Paul Ogilvie, Yi Zhang, and Jamie Callan. Information filtering, novelty detection, and named-page finding. In *Proceedings of the 11th Text Retrieval Conference*, 2002.

[46] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, 2011.

[47] David Cossock and Tong Zhang. Subset ranking using regression. In *Proceedings of the 19th Annual Conference on Learning Theory*, page 605–619. Springer-Verlag, 2006.

[48] Eric Crestan and Patrick Pantel. A fine-grained taxonomy of tables on the web. In *CIKM*, 2010.

[49] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3079–3087, 2015.

[50] 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*, 2019.

[51] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proc. of the 11th ACM Int'l Conf. on Web Search and Data Mining (WSDM)*, page 126–134, 2018.

[52] Nilesh N. Dalvi, Vibhor Rastogi, Anirban Dasgupta, Anish Das Sarma, and Tamás Sarlós. Optimal hashing schemes for entity matching. In *22nd International World Wide Web Conference, WWW*, pages 295–306, 2013.

[53] 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.

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

[55] Aryan Deshwal, Janardhan Rao Doppa, and Dan Roth. Learning and inference for structured prediction: A unifying perspective. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial*

Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, pages 6291–6299. ijcai.org, 2019.

[56] Aryan Deshwal, Janardhan Rao Doppa, and Dan Roth. Learning and inference for structured prediction: A unifying perspective. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6291–6299. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[57] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *32nd AAAI Conference on Artificial Intelligence*, pages 1811–1818, 2018.

[58] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[59] T. Dharani and I. L. Aroquiaraj. A survey on content based image retrieval. In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pages 485–490, 2013.

[60] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.

[61] J. Eberius, K. Braunschweig, M. Hentsch, M. Thiele, A. Ahmadov, and W. Lehner. Building the dresden web table corpus: A classification approach. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 41–50, 2015.

[62] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.*, 11(11):1454–1467, 2018.

[63] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. Matching web tables with knowledge base entities: From entity

lookups to entity embeddings. In *The Semantic Web – ISWC 2017*, pages 260–277, Cham, 2017. Springer International Publishing.

[64] Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. NADEEF/ER: generic and interactive entity resolution. In *International Conference on Management of Data, SIGMOD 2014*, pages 1071–1074. ACM, 2014.

[65] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[66] Jeffrey L. Elman. Finding structure in time. *Cogn. Sci.*, 14(2):179–211, 1990.

[67] Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, and Xueqi Cheng. Modeling diverse relevance patterns in ad-hoc retrieval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 375–384. ACM, 2018.

[68] Jeffrey Fisher, Peter Christen, Qing Wang, and Erhard Rahm. A clustering-based framework to control block sizes for entity resolution. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 279–288. ACM, 2015.

[69] Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 170–178. Morgan Kaufmann, 1998.

[70] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. End-to-end multi-perspective matching for entity resolution. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 4961–4967. ijcai.org, 2019.

[71] Norbert Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Trans. Inf. Syst.*, 7(3):183–204, 1989.

[72] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1183–1192.

[73] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(4):744–755, 2014.

[74] Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritze, and Heiko Paulheim. Entity matching on web tables: a table embeddings approach for blocking. In *EDBT*, 2017.

[75] Fredric C. Gey. Inferring probability of relevance using the method of logistic regression. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, pages 222–231. ACM/Springer, 1994.

[76] Majid Ghasemi-Gol and Pedro A. Szekely. Tabvec: Table vectors for classification of web tables. *CoRR*, abs/1802.06290, 2018.

[77] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 1263–1272. JMLR.org, 2017.

[78] Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, and Warren Shen. Google fusion tables: data management, integration and collaboration in the cloud. In *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pages 175–180. ACM, 2010.

[79] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE, 2013.

[80] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 55–64. ACM, 2016.

[81] Jiang Guo, Darsh J. Shah, and Regina Barzilay. Multi-source domain adaptation with mixture of experts. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4694–4703. Association for Computational Linguistics, 2018.

[82] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3887–3896. PMLR, 2020.

[83] Suchin Gururangan, Ana Marasovic, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020*, pages 8342–8360. Association for Computational Linguistics, 2020.

[84] Michael U. Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *J. Mach. Learn. Res.*, 13(1):307–361, February 2012.

[85] Maryam Habibi, Johannes Starlinger, and Ulf Leser. Tabsim: A siamese neural network for accurate estimation of table similarity. *CoRR*, abs/2008.10856, 2020.

[86] Xiaochuang Han and Jacob Eisenstein. Unsupervised domain adaptation of contextualized embeddings for sequence labeling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019*, pages 4237–4247. Association for Computational Linguistics, 2019.

[87] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[88] 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. ACM, 2017.

[89] Hua He and Jimmy J. Lin. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 937–948. The Association for Computational Linguistics, 2016.

[90] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, 88, 01 2000.

[91] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. Open domain question answering over tables via dense retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 512–519. Association for Computational Linguistics, 2021.

[92] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4320–4333. Association for Computational Linguistics, 2020.

[93] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[94] Sebastian Hofstätter, Hamed Zamani, Bhaskar Mitra, Nick Craswell, and Allan Hanbury. Local self-attention over long text for efficient document retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 2021–2024, New York, NY, USA, 2020. Association for Computing Machinery.

[95] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2042–2050, 2014.

[96] Jim C. Huang and Brendan J. Frey. Structured ranking learning using cumulative distribution networks. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 697–704. Curran Associates, Inc., 2008.

[97] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 2333–2338. ACM, 2013.

[98] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015.

[99] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. PACRR: A position-aware neural IR model for relevance matching. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1049–1058. Association for Computational Linguistics, 2017.

[100] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. Co-pacrr: A context-aware neural IR model for ad-hoc retrieval. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 279–287. ACM, 2018.

[101] 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, page 1500–1508. Association for Computing Machinery, 2019.

[102] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613. ACM, 1998.

[103] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37, pages 448–456. JMLR.org, 2015.

[104] Aaron Jaech, Hetunandan Kamisetty, Eric K. Ringger, and Charlie Clarke. Match-tensor: a deep relevance model for search. *ArXiv*, abs/1701.07795, 2017.

[105] Anuj Jaiswal, David J. Miller, and Prasenjit Mitra. Schema matching and embedded value mapping for databases with opaque column names and mixed continuous and discrete-valued data fields. *ACM Trans. Database Syst.*, 38(1), April 2013.

[106] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

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

[108] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

[109] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 133–142. ACM, 2002.

[110] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547, 2021.

[111] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014.

[112] Jaewoo Kang and Jeffrey F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2003.

[113] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):664–676, 2017.

[114] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. Low-resource deep entity resolution with transfer and active learning. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5851–5861. Association for Computational Linguistics, 2019.

[115] Tom Kenter and Maarten de Rijke. Short text similarity with word embeddings. In *CIKM*, 2015.

[116] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 39–48, New York, NY, USA, 2020. Association for Computing Machinery.

[117] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[118] Yoon Kim, Yacine Jernite, David A. Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2741–2749. AAAI Press, 2016.

[119] Young-Bum Kim, Karl Stratos, and Dongchan Kim. Domain attention with an ensemble of experts. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017*, pages 643–653. Association for Computational Linguistics, 2017.

[120] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

[122] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[123] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. Magellan: Toward building entity matching management systems. *Proc. VLDB Endow.*, 9(12):1197–1208, 2016.

[124] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3(1):484–493, 2010.

[125] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 4364–4373. Association for Computational Linguistics, 2019.

[126] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[127] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 282–289. Morgan Kaufmann, 2001.

[128] Christoph H. Lampert and Matthew B. Blaschko. Structured prediction by joint kernel support estimation. *Mach. Learn.*, 77(2-3):249–269, 2009.

[129] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 260–270. The Association for Computational Linguistics, 2016.

[130] Wuwei Lan and Wei Xu. Character-based neural networks for sentence pair modeling. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 157–163. Association for Computational Linguistics, 2018.

[131] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, pages 1188–1196, 2014.

[132] Yann LeCun and Yoshua Bengio. *Convolutional Networks for Images, Speech, and Time Series*, page 255–258. MIT Press, Cambridge, MA, USA, 1998.

[133] E. L. Lehmann and Joseph P. Romano. *Testing statistical hypotheses.* Springer Texts in Statistics. Springer, New York, 2005.

[134] 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*, 6:167–195, 2015.

[135] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. The mannheim search join engine. *Web Semant.*, 35(P3):159–166, 2015.

[136] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *52nd ACL*, pages 302–308, 2014.

[137] Ping Li, Christopher J. C. Burges, and Qiang Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 897–904. Curran Associates, Inc., 2007.

[138] Quanzhi Li, Sameena Shah, and Rui Fang. Table classification using both structure and content information: A case study of financial documents. In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 1778–1783. IEEE Computer Society, 2016.

[139] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.*, 14(1):50–60, 2020.

[140] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1):1338–1347, 2010.

[141] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1–2):1338–1347, September 2010.

[142] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, pages 2181–2187, 2015.

[143] Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. Table-to-text generation by structure-aware seq2seq learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI*

Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 4881–4888. AAAI Press, 2018.

[144] Tie-Yan Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, March 2009.

[145] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, July 2019.

[146] Xiaoyong Liu and W. Bruce Croft. Passage retrieval based on language models. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, page 375–382, New York, NY, USA, 2002. Association for Computing Machinery.

[147] Ying Liu, Kun Bai, Prasenjit Mitra, and C. Lee Giles. Tablerank: A ranking algorithm for table search and retrieval. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 317–322. AAAI Press, 2007.

[148] 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*, abs/1907.11692, 2019.

[149] Jiaheng Lu, Chunbin Lin, Jin Wang, and Chen Li. Synergy of database techniques and machine learning models for string similarity search and join. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019*, pages 2975–2976. ACM, 2019.

[150] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 289–297, 2016.

[151] Xiaofei Ma, Peng Xu, Zhiguo Wang, Ramesh Nallapati, and Bing Xiang. Domain adaptation with bert-based domain classification and data selection. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP*, pages 76–83. Association for Computational Linguistics, 2019.

[152] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[153] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. CEDR: contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 1101–1104. ACM, 2019.

[154] Erin MacDonald and Denilson Barbosa. Neural relation extraction on wikipedia tables for augmenting knowledge graphs. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 2133–2136. ACM, 2020.

[155] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, USA, 2008.

[156] 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. *CoRR*, abs/1908.06780, 2019.

[157] Ryan T. McDonald, George Brokos, and Ion Androutsopoulos. Deep relevance ranking using enhanced document-query interactions. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1849–1860. Association for Computational Linguistics, 2018.

[158] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference*

on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013.

[159] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, pages 3111–3119, 2013.

[160] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[161] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web, WWW*, pages 1291–1299, 2017.

[162] Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning, ICML '07*, pages 641–648. ACM, 2007.

[163] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *ICML*, 2012.

[164] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. Using linked data to mine rdf from wikipedia's tables. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM '14*, pages 533–542, 2014.

[165] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 19–34. ACM, 2018.

[166] J. Mueller and A. Smola. Recognizing variables from their data via deep embeddings of distributions. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1264–1269, 2019.

[167] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2786–2792. AAAI Press, 2016.

[168] Tam Nguyen, Quoc Viet Hung Nguyen, Matthias Weidlich, and Karl Aberer. Result selection and summarization for web table search. *Proceedings - International Conference on Data Engineering*, 2015:231–242, 05 2015.

[169] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016.

[170] Yifan Nie, Yanling Li, and Jian-Yun Nie. Empirical study of multi-level convolution models for IR based on representations and interactions. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval, ICTIR 2018, Tianjin, China, September 14-17, 2018*, pages 59–66. ACM, 2018.

[171] Kyosuke Nishida, Kugatsu Sadamitsu, Ryuichiro Higashinaka, and Yoshihiro Matsuo. Understanding the semantic structures of tables with a hybrid deep neural network architecture. In *AAAI*, 2017.

[172] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *ArXiv*, abs/1901.04085, 2019.

[173] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-stage document ranking with bert. *ArXiv*, abs/1910.14424, 2019.

[174] Sebastian Nowozin and Christoph H. Lampert. Structured learning and prediction in computer vision. *Found. Trends Comput. Graph. Vis.*, 6(3-4):185–365, 2011.

[175] Paul Ogilvie, Jamie Callan, 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 Informaion Retrieval, SIGIR '03*, pages 143–150. ACM, 2003.

[176] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 24(4):694–707, April 2016.

[177] Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010*, pages 751–760. ACM, 2010.

[178] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2793–2799. AAAI Press, 2016.

[179] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, page 257–266, 2017.

[180] Patrick Pantel and Eric Crestan. Web-scale table census and classification. In *WSDM*, 2011.

[181] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2):31:1–31:42, 2020.

[182] Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2249–2255. The Association for Computational Linguistics, 2016.

[183] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013.

[184] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[185] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, October 2014.

[186] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[187] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1756–1765. Association for Computational Linguistics, 2017.

[188] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018.

[189] Minh Pham, Suresh Alse, Craig A. Knoblock, and Pedro A. Szekely. Semantic labeling: A domain-independent approach. In *International Semantic Web Conference*, 2016.

[190] Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. *Proc. VLDB Endow.*, 5(10):908–919, June 2012.

[191] R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975.

[192] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '98*, pages 275–281. ACM, 1998.

[193] Tao Qin, Tie-Yan Liu, and Hang Li. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval*, 13:375–397, 2009.

[194] Xipeng Qiu and Xuanjing Huang. Convolutional neural tensor network architecture for community-based question answering. In *Proceedings of the 24th International Conference on Artificial Intelligence*, page 1305–1311. AAAI Press, 2015.

[195] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5835–5847. Association for Computational Linguistics, 2021.

[196] Ariadna Quattoni, Arnau Ramisa, Pranava Swaroop Madhyastha, Edgar Simo-Serra, and Francesc Moreno-Noguer. Structured prediction with output embeddings for semantic image annotation. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 552–557. The Association for Computational Linguistics, 2016.

[197] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[198] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, page 381–390. Morgan Kaufmann Publishers Inc., 2001.

[199] S. K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro A. Szekely. Assigning semantic labels to data sources. In *ESWC*, 2015.

[200] Alexander Rietzler, Sebastian Stabinger, Paul Opitz, and Stefan Engl. Adapt or get left behind: Domain adaptation through BERT language model finetuning for aspect-target sentiment classification. In *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020*, pages 4933–4941. European Language Resources Association, 2020.

[201] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

[202] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST), 1994.

[203] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. Reasoning about Entailment with Neural Attention. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[204] Natalia Ruemmele, Yuriy Tyshetskiy, and Alex Collins. Evaluating approaches for supervised semantic labeling. In *TheWebConf Workshop: Linked Data on the Web (LDOW)*, 01 2018.

[205] 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. Association for Computing Machinery, 2019.

[206] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[207] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 269–278. ACM, 2002.

[208] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *Lecture Notes in Computer Science*, page 593–607, 2018.

[209] Yoones A. Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. Knowledge base augmentation using tabular data. In *LDOW*, 2014.

[210] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[211] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18)*, pages 4058–4065.

[212] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Trans. Knowl. Data Eng.*, 27(2):443–460, 2015.

[213] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 101–110. ACM, 2014.

[214] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search.

In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 373–374. ACM, 2014.

[215] 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.

[216] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[217] Deepika Singh, Erinc Merdivan, Ismini Psychoula, Johannes Kropf, Sten Hanke, Matthieu Geist, and Andreas Holzinger. Human activity recognition using recurrent neural networks. In *Machine Learning and Knowledge Extraction*, pages 267–274, Cham, 2017. Springer International Publishing.

[218] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Synthesizing entity matching rules by examples. *Proc. VLDB Endow.*, 11(2):189–202, 2017.

[219] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013.

[220] Serena Sorrentino, Sonia Bergamaschi, and Maciej Gawinecki. NORMS: an automatic tool to perform schema label normalization. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1344–1347. IEEE Computer Society, 2011.

[221] Serena Sorrentino, Sonia Bergamaschi, Maciej Gawinecki, and Laura Po. Schema label normalization for improving schema matching. *Data Knowl. Eng.*, 69(12):1254–1273, 2010.

[222] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW '07*, 2007.

[223] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016*, pages 2058–2065. AAAI Press, 2016.

[224] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In Maosong Sun, Xuanjing Huang, Heng Ji, Zhiyuan Liu, and Yang Liu, editors, *Chinese Computational Linguistics*, pages 194–206, Cham, 2019. Springer International Publishing.

[225] 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, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 771–782. ACM, 2016.

[226] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, page 1017–1024, Madison, WI, USA, 2011. Omnipress.

[227] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1017–1024. Omnipress, 2011.

[228] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

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

[230] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015.

[231] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society, 2016.

[232] Chuanqi Tan, Furu Wei, Wenhui Wang, Weifeng Lv, and Ming Zhou. Multiway attention networks for modeling sentence pairs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4411–4417. ijcai.org, 2018.

[233] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1422–1432. The Association for Computational Linguistics, 2015.

[234] Zhiwen Tang and Grace Hui Yang. Deeptilebars: Visualizing term distribution for neural information retrieval. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 289–296. AAAI Press, 2019.

[235] Mohamed Trabelsi, Jin Cao, and Jeff Heflin. Semantic labeling using a deep contextualized language model. *CoRR*, abs/2010.16037, 2020.

[236] Mohamed Trabelsi, Jin Cao, and Jeff Heflin. Selab: Semantic labeling with bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[237] Mohamed Trabelsi, Zhiyu Chen, Brian D. Davison, and Jeff Heflin. A hybrid deep model for learning to rank data tables. In *2020 IEEE International Conference on Big Data (Big Data)*, 2020.

[238] 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 2020, Atlanta, GA, USA, December 10-13, 2020, pages 3005–3014. IEEE, 2020.

[239] Mohamed Trabelsi, Zhiyu Chen, Brian D. Davison, and Jeff Heflin. Neural ranking models for document retrieval. *Inf. Retr. J.*, 24(6):400–444, 2021.

[240] 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 Web Conference (WWW 2022)*, 2022.

[241] Mohamed Trabelsi, Brian D. Davison, and Jeff Heflin. Improved table retrieval using multiple context embeddings for attributes. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1238–1244, 2019.

[242] Mohamed Trabelsi, Jeff Heflin, and Jin Cao. Dame: Domain adaptation for matching entities. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining (WSDM 2022)*, 2022.

[243] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, page 2071–2080. JMLR.org, 2016.

[244] Isabel Valera and Zoubin Ghahramani. Automatic discovery of the statistical types of variables in a dataset. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3521–3529. PMLR, 06–11 Aug 2017.

[245] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014.

[246] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. 2017.

[247] 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.

[248] 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.

[249] Maksims Volkovs and Richard S. Zemel. Boltzrank: learning to maximize expected ranking gain. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 1089–1096. ACM, 2009.

[250] Ji Wan, P. Wu, S. Hoi, P. Zhao, Xingyu Gao, Dayong Wang, Yongdong Zhang, and J. Li. Online learning to rank for content-based image retrieval. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, page 2284–2290, 2015.

[251] Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. A deep architecture for semantic matching with multiple positional sentence representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2835–2841. AAAI Press, 2016.

[252] Shengxian Wan, Yanyan Lan, Jun Xu, Jiafeng Guo, Liang Pang, and Xueqi Cheng. Match-srnn: Modeling the recursive matching structure with spatial rnn. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, page 2922–2928. AAAI Press, 2016.

[253] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, November 2018.

[254] Baiyang Wang and Diego Klabjan. An attention-based deep net for learning to rank. *ArXiv*, abs/1702.06106, 2017.

[255] Dan Wang and Yi Shang. A new active labeling method for deep learning. In *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pages 112–119. IEEE, 2014.

[256] Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro A. Szekely. Retrieving complex tables with multi-granular graph representation learning. In *The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1472–1482. ACM, 2021.

[257] Shuohang Wang and Jing Jiang. A compare-aggregate model for matching text sequences. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net, 2017.

[258] Wenya Wang, Sinno Jialin Pan, Daniel Dahlmeier, and Xiaokui Xiao. Coupled multi-layer attentions for co-extraction of aspect and opinion terms. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 3316–3322. AAAI Press, 2017.

[259] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*, pages 1112–1119, 2014.

[260] Ross Wilkinson. Effective retrieval of structured documents. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, pages 311–317. ACM/Springer, 1994.

[261] Dustin Wright and Isabelle Augenstein. Transformer based multi-source domain adaptation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*, pages 7963–7974. Association for Computational Linguistics, 2020.

[262] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '17*, pages 55–64. ACM, 2017.

[263] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proc. 40th Int'l ACM SIGIR Conf. on Research and Development in Information Retr.*, pages 55–64, 2017.

[264] 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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[265] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Product knowledge graph embedding for e-commerce. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 672–680, 2020.

[266] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD Conference*, 2012.

[267] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 97–108. ACM, 2012.

[268] Zhao Yan, Duyu Tang, Nan Duan, Jun-Wei Bao, Yuanhua Lv, Ming Zhou, and Zhoujun Li. Content-based table retrieval for web queries. *Neurocomputing*, 349:183–189, 2017.

[269] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, May 2015.

[270] Liu Yang, Qingyao Ai, Jiafeng Guo, and W. Bruce Croft. anmm: Ranking short answer texts with attention-based neural matching model. *Proceedings*

of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16, 2016.

[271] Wei Yang, Haotian Zhang, and Jimmy Lin. Simple applications of bert for ad hoc document retrieval. *ArXiv*, abs/1903.10972, 2019.

[272] Zhilin Yang, Ruslan Salakhutdinov, and William W. Cohen. Transfer learning for sequence tagging with hierarchical recurrent networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[273] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical attention networks for document classification. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1480–1489. The Association for Computational Linguistics, 2016.

[274] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical attention networks for document classification. In *Proc. of Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[275] Yang Yi, Zhiyu Chen, Jeff Heflin, and Brian D. Davison. Recognizing quantity names for tabular data. In *Joint Proceedings of the First International Workshop on Professional Search (ProfS2018); the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding (KG4IR); and the International Workshop on Data Search (DATA:SEARCH'18) Co-located with (ACM SIGIR 2018), Ann Arbor, Michigan, USA, July 12, 2018*, volume 2127 of *CEUR Workshop Proceedings*, pages 68–73. CEUR-WS.org, 2018.

[276] Zeynep Akkalyoncu Yilmaz, Shengjin Wang, Wei Yang, Haotian Zhang, and Jimmy Lin. Applying BERT to document retrieval with birch. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing,*

EMNLP-IJCNLP 2019, pages 19–24. Association for Computational Linguistics, 2019.

[277] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426. Association for Computational Linguistics, July 2020.

[278] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272, 2015.

[279] Haochao Ying, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. Sequential recommender system based on hierarchical attention networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 3926–3932. ijcai.org, 2018.

[280] Minoru Yoshida and Kentaro Torisawa. A method to integrate tables of the world wide web. In *In Proceedings of the International Workshop on Web Document Analysis (WDA 2001*, pages 31–34, 2001.

[281] S. Yu, J. Su, and D. Luo. Improving bert-based text classification with auxiliary sentence and domain knowledge. *IEEE Access*, 7:176600–176612, 2019.

[282] Benjamin Zapilko, Matthäus Zloch, and Johann Schaible. Utilizing regular expressions for instance-based schema matching. In *Proceedings of the 7th International Conference on Ontology Matching - Volume 946*, page 240–241, 2012.

[283] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '01*, pages 334–342. ACM, 2001.

[284] ChengXiang Zhai, ChengXiang Zhai, and John Lafferty. Two-stage language models for information retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '02*, pages 49–56. ACM, 2002.

[285] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Jointly optimizing query encoder and product quantization to improve retrieval performance. *CoRR*, abs/2108.00644, 2021.

[286] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing dense retrieval model training with hard negatives. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 1503–1512. ACM, 2021.

[287] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. Repbert: Contextualized text embeddings for first-stage retrieval. *CoRR*, abs/2006.15498, 2020.

[288] Haotian Zhang, Mustafa Abualsaud, Nimesh Ghelani, Mark D. Smucker, Gordon V. Cormack, and Maura R. Grossman. Effective user interaction for high-recall retrieval: Less is more. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, page 187–196, 2018.

[289] Kun Zhang, Guangyi Lv, Linyuan Wang, Le Wu, Enhong Chen, Fangzhao Wu, and Xing Xie. Drr-net: Dynamic re-read network for sentence semantic matching. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 7442–7449. AAAI Press, 2019.

[290] Li Zhang, Shuo Zhang, and Krisztian Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1029–1032, New York, NY, USA, 2019. Association for Computing Machinery.

[291] Shuo Zhang and K. Balog. Recommending related tables. *ArXiv*, abs/1907.03595, 2019.

[292] Shuo Zhang and Krisztian Balog. Design patterns for fusion-based object retrieval. In *Advances in Information Retrieval - 39th European Conference on IR Research (ECIR)*, pages 684–690, April 2017.

[293] Shuo Zhang and Krisztian Balog. Ad hoc table retrieval using semantic similarity. In *WWW*, 2018.

[294] Shuo Zhang and Krisztian Balog. Auto-completion for data cells in relational tables. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 761–770. ACM, 2019.

[295] Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. Novel entity discovery from web tables. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 1298–1308. ACM / IW3C2, 2020.

[296] Yuan Zhang, Regina Barzilay, and Tommi S. Jaakkola. Aspect-augmented adversarial networks for domain adaptation. *Trans. Assoc. Comput. Linguistics*, 5:515–528, 2017.

[297] Chen Zhao and Yeye He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference, WWW 2019*, pages 2413–2424. ACM, 2019.

[298] Z. Zheng, H. Zha, and G. Sun. Query-level learning to rank using isotonic regression. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 1108–1115, Sep. 2008.

[299] Zhaohui Zheng, Keke Chen, Gordon Sun, and Hongyuan Zha. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*, pages 287–294. ACM, 2007.

[300] W. Zhou, H. Li, and Q. Tian. Recent advance in content-based image retrieval: A literature survey. *ArXiv*, abs/1706.06064, 2017.

Vita

Mohamed Trabelsi is from Sfax, Tunisia. He completed his Ph.D. in Computer Science from Lehigh University, PA, USA in May, 2022. He holds a Master of Science in Computer Science from University of Louisville, KY, USA. He also holds a Bachelor of Science in Computer Science from Tunisia Polytechnic School, Tunisia.

LIST OF PUBLICATIONS

M Trabelsi, Z Chen, S Zhang, BD Davison, J Heflin "StruBERT: Structure-aware BERT for Table Search and Matching". *Proceedings of the Web Conference (WWW'22)*, April 2022.

M Trabelsi, J Heflin, J Cao "DAME: Domain Adaptation for Matching Entities". *Proceedings of the 15th ACM International Conference on Web Search and Data Mining (WSDM 2022)*, February 2022.

Z Chen, **M Trabelsi**, J Heflin, D Yin, BD Davison "MGNETS: Multi-Graph Neural Networks for Table Search". *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM 2021)*, November 2021.

M Trabelsi, Z Chen, BD Davison, J Heflin "Neural ranking models for document retrieval". *Information Retrieval Journal*, 24(6): 400-444, 2021.

M Trabelsi, J Cao, J Heflin "SeLaB: Semantic Labeling with BERT". *International Joint Conference on Neural Networks (IJCNN)*, 1-8, 2021.

M Trabelsi, Z Chen, BD Davison, J Heflin "A Hybrid Deep Model for Learning to Rank Data Tables". *IEEE International Conference on Big Data (Big Data)*, 979-986, 2020.

M Trabelsi, Z Chen, BD Davison, J Heflin "Relational Graph Embeddings for Table Retrieval". *IEEE International Conference on Big Data (Big Data)*, 3005-3014, 2020.

M Trabelsi, J Cao, J Heflin "Semantic Labeling Using a Deep Contextualized Language Model". *CoRR abs/2010.16037*, 2020.

Z Chen, **M Trabelsi**, J Heflin, Y Xu, BD Davison "Table Search Using a Deep Contextualized Language Model". *Proceedings of 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 589-598, 2020.

Z Chen, **M Trabelsi**, BD Davison, J Heflin "Towards Knowledge Acquisition of Metadata on AI Progress". *ISWC (Posters and Demos)*, 232-237, 2020.

M Trabelsi, BD Davison, J Heflin "Improved table retrieval using multiple context embeddings for attributes". *IEEE International Conference on Big Data (Big Data)*, 1238-1244, 2019.

M Trabelsi, H Frigui "Robust fuzzy clustering for multiple instance regression". *Pattern Recognition*, 90: 424-435, 2019.

M Trabelsi, H Frigui "Fuzzy and possibilistic clustering for multiple instance linear regression". *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 1-7, 2018.

A Karem, **M Trabelsi**, M Moalla, H Frigui "Comparison of several single and multiple instance learning methods for detecting buried explosive objects using GPR data". *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XXIII*, 2018.