Multiple-Choice Question Answering Over Semi-Structured Tables

by

Weite Ni

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Weite Ni, 2019

Abstract

Question answering (QA) is the task of automatically finding answers to natural language questions. A QA system requires access to some form of knowledge in order to find the answers. Most QA tasks use raw text corpora or structured knowledge bases as knowledge. However, raw text corpora, although easy to get in large quantities, are hard to reason with by machines. Structured knowledge bases are easy to reason with, but require manual effort to normalize. We view semi-structured tables as a compromise between raw text corpora and structured knowledge bases. Semi-structured tables require less manual effort to build comparing with structured knowledge bases, and their structured properties make it easy for automated reasoning.

In this thesis, we build a QA system that can answer multiple-choice questions based on semi-structured tables. We tackle the task in two steps: table retrieval and answer selection. To retrieve the most relevant table to the questions, we build a feature-based model that can effectively take the candidate choices into account. To find the best answer based on the retrieved table, we first measure the relevance between the question and rows in the table, then extract the best answer from the most relevant rows. Evaluation on the TabMCQ benchmark shows that our system achieves a huge improvement over the previous state-of-the-art system.

We are all in the gutter, but some of us are looking at the stars.

– Oscar Wilde.

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor Denilson Barbosa for his advice and support. When I felt lost in my research, he would point out possible directions I could go. When I felt depressed, he would cheer me up. I would not be able to finish this thesis without his patience and support. I'm also very grateful to my examining committee members, Abram Hindle and Nilanjan Ray, for their valuable advice.

I would also like to thank my girl friend Ruijing Han for her support and love during the past four years. She is always there when I need her. My thanks also go to all my friends, who have made my life in Edmonton colorful.

Last but not least, I would like to thank my parents, who help me and encourage me throughout my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

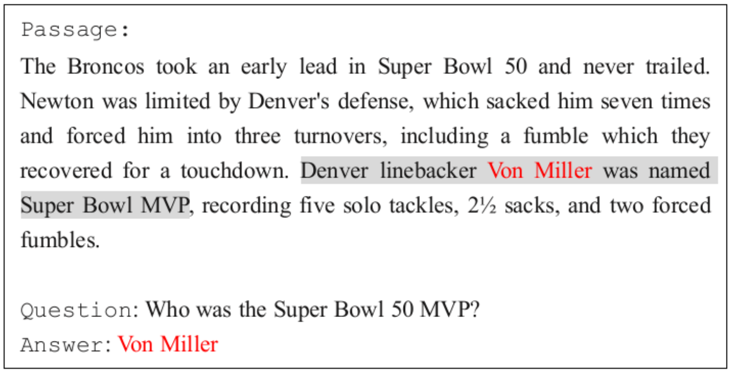
Question answering (QA) is the task of automatically finding answers to natural language questions. It has a long history dating back to the 1960s [5]. Researchers from many different fields, including information retrieval (IR) and natural language processing (NLP), have shown considerable interests in this area. Question answering systems have a variety of real-life applications, like search engines and chatbots, and other conversational agents.

There are many types of questions, He et al. [6] divided them into the types described in Table 1.1. Factoid questions are answered with simple facts, like an entity, a description or simply yes/no. Usually, a factoid question can be answered with short text and has a definite answer. Opinion questions ask about subjective opinions. Such questions do not have definite answers, and it's difficult to assess the quality of answers.

	Factoid	**Opinion**
Entity	What is the capital of Canada?	Top 10 movies of 2018
Description	Why is the sky blue?	How good is Toyota Carola?
YesNo	Was Abraham Lincoln American?	Does playing go improve intelligence?

Table 1.1: Different types of questions.

Most question answering systems focus on **factoid questions**, especially those can be answered with entities. Jurafsky and Martin [12] divided the



Passage:

The Broncos took an early lead in Super Bowl 50 and never trailed. Newton was limited by Denver's defense, which sacked him seven times and forced him into three turnovers, including a fumble which they recovered for a touchdown. Denver linebacker Von Miller was named Super Bowl MVP, recording five solo tackles, 2½ sacks, and two forced fumbles.

Question: Who was the Super Bowl 50 MVP?

Answer: Von Miller

Figure 1.1: An example from an IR-based QA dataset. The sentence in grey is the sentence that has the answer to the question.

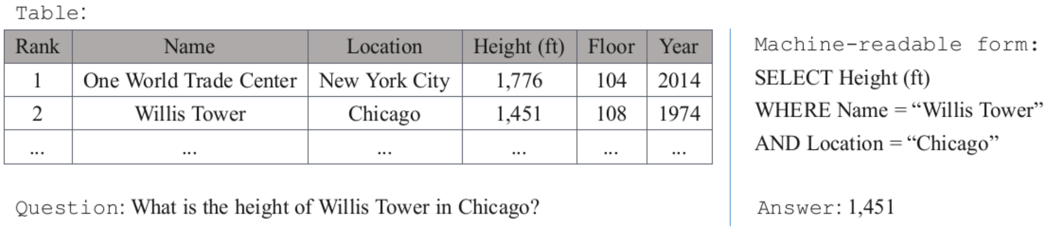


Table:

Rank	Name	Location	Height (ft)	Floor	Year
1	One World Trade Center	New York City	1,776	104	2014
2	Willis Tower	Chicago	1,451	108	1974
...	...	...	...	...	...

Question: What is the height of Willis Tower in Chicago?

Machine-readable form:
SELECT Height (ft)
WHERE Name = "Willis Tower"
AND Location = "Chicago"

Answer: 1,451

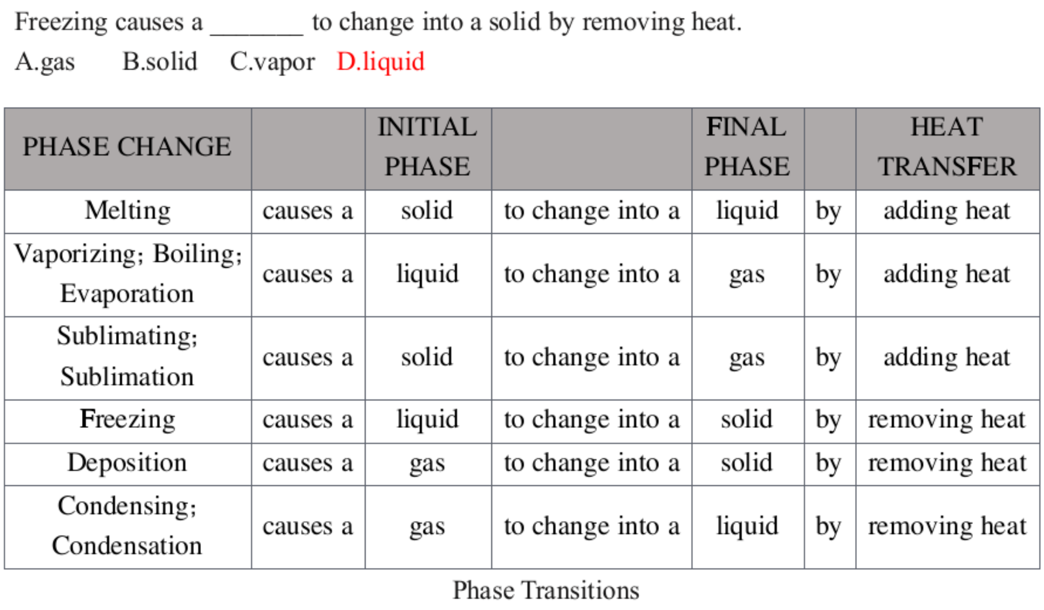
Figure 1.2: An example from a knowledge-based QA dataset. There's no need to predict FROM clause because it's assumed that the relevant table is given.

systems for answering factoid questions into two categories: **information-retrieval or IR-based** and **knowledge-based**.

IR-based QA relies on textual information like documents, and the answer is typically a text span in a specific document [23]. Given a question, the system first needs to find the relevant documents or passages, and then uses reading comprehension techniques to extract the answer from the text. Figure 1.1 shows an example from a popular IR-based QA dataset, SQuAD [20].

Knowledge-based QA focuses on semantic parsing, i.e., translating a natural language question into a machine-readable form. Then the machine-readable form is used to query a structured database to get the answer. Figure 1.2 shows an example from WikiSQL dataset [42], which aims at mapping questions to SQL queries [3].

In this thesis, we focus on IR-based question answering, and the questions are answered with entities. A complete IR-based QA system usually consists of three stages: question processing, document retrieval and ranking, and an-



Freezing causes a _______ to change into a solid by removing heat.

A.gas B.solid C.vapor D.liquid

PHASE CHANGE		INITIAL PHASE		FINAL PHASE		HEAT TRANSFER
Melting	causes a	solid	to change into a	liquid	by	adding heat
Vaporizing; Boiling; Evaporation	causes a	liquid	to change into a	gas	by	adding heat
Sublimating; Sublimation	causes a	solid	to change into a	gas	by	adding heat
Freezing	causes a	liquid	to change into a	solid	by	removing heat
Deposition	causes a	gas	to change into a	solid	by	removing heat
Condensing; Condensation	causes a	gas	to change into a	liquid	by	removing heat

Phase Transitions

Figure 1.3: A multiple-choice question over a semi-structured table.

swer extraction [12]. The question processing stage is done before sending the question to the IR system. Query formulation and answer type detection are commonly used processing techniques. Document retrieval aims at retrieving documents that are relevant to the question. The final stage, answer extraction, is to extract the answer from the relevant documents.

Although the frameworks of different IR-based QA systems are similar, it is difficult to build a system that works for every question answering task. The textual information the system relies on could be documents, web, or tables, and the strategies to deal with them are different. Also, sometimes the question is based on only one document, sometimes it requires inference from several documents. In this thesis, we focus on a specific case: multiple-choice (MC) question answering over semi-structured tables. Figure 1.3 gives an example. Each row of a semi-structured table is a sentence stating a fact. And the tables are not normalized. Some cells may contain multiple parts like "*Sublimating; Sublimation*", some cells can be blank. And some columns may not have a name.

1.2 Motivation

We are motivated by the special properties of semi-structured tables. Raw text corpora, like the passage in Figure 1.1, contain no structure, and are hard to reason with by machines. But they are easy to get in large quantities. Structured knowledge, like the structured table in Figure 1.2, on the other hand, although easy to reason with, require manual effort to normalize. Semi-structured tables, like the table in Figure 1.3, can be viewed as "a compromise in the trade-off between degree of structure and ubiquity" [9]. So we would like to use semi-structured tables as source knowledge for our QA system.

Following the general framework of IR-based QA system, to solve the problem of multiple-choice question answering over semi-structured tables, the system should contain three stages: question processing, **table** retrieval, and answer **selection**. We notice that table retrieval is usually omitted in QA datasets with tables as knowledge, like WikiSQL [42] and WikiTableQuestions [15]. It's assumed that the relevant table is given, which is not always the case when it comes to real-life applications. So another motivation is that we want to explore different ways to do table retrieval.

1.3 Overview

The state-of-the-art method on the task is proposed by Wang et al. [31]. They tackle this task with two steps: table retrieval and answer selection. In the first step, they use a neural model to rank the tables according to relevance to the question, and retrieve several most relevant tables. In the second step, they first compute a relevance score for every cell in the tables, and then match the candidate choices with the most relevant cells to find the correct answer. In this thesis, we build our QA system based on Wang et al.'s method. Figure 1.4 shows the overview of our QA system.

In both table retrieval and answer selection, Wang et al. only use the distributed representations of words, and focus on semantic similarity. However, models based on distributed representations of words are not sensitive to

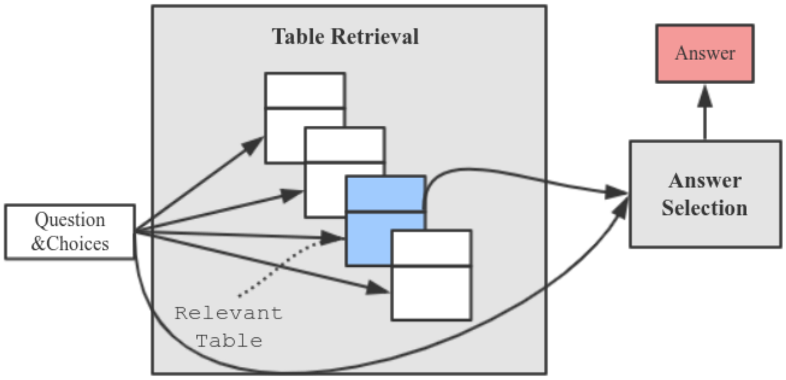


Figure 1.4: Overview of our system.

numbers and proper nouns. And dealing with numbers and proper nouns is very important for factoid question answering [26]. So we also consider lexical similarity in our system.

1.4 Thesis Statement and Contributions

In this thesis, we focus on answering two questions. For table retrieval, can a feature-based model that focuses on lexical similarity perform comparably well with a neural model? For answer selection, can we improve the performance of the neural model by taking lexical similarity into account?

There are three major contributions in this work:

First, we design a feature-based model to do table retrieval, and propose two novel features. Wang et al.'s table retrieval model is a neural model based on semantic representations of words. However, because of the difficulty in semantically representing the cells in the table, their model doesn't take the table cells into account. And they also fail to use the candidate choices as additional information to do table retrieval. Our model, mainly based on lexical similarity, is more flexible, and can efficiently utilize the candidate choices and table cells.

Second, in the stage of answer selection, we augment their method by filtering out irrelevant cells. In order to find the correct answer, Wang et al. score every cell in the table. However, not all cells are relevant to the candidate choices. For example, a question about the table in Figure 1.3 is "*How to turn a solid into liquid?*", and the candidate choices are "*A. adding heat; B. removing heat*", the cells in the first column are totally irrelevant, and it's not necessary to score those cells. So we use a simple string matching method to filter out irrelevant cells and reduce noise.

Third, we improve their cell-scoring method by adding lexical features. Wang et al. score the cells based on semantic representations of words, and we believe that the combination of lexical and semantic matching will be better than semantic alone.

Evaluation on the TabMCQ [10] benchmark shows that our modifications improve the accuracy of Wang et al.'s model from 79.0% to 91.9%, which is a 16.3% relative increase. The third modification, adding lexical features to cell-scoring method, results in the most significant absolute improvement of 6.9%.

1.5 Organization of the Thesis

Chapter 2 provides an overview of related work. In Chapter 3, we describe our feature-based table retrieval model, and Wang et al.'s semantic-based model. In Chapter 4, we present our answer selection method. In Chapter 5, we evaluate our models and present the results. We conclude the thesis and propose works for future research in Chapter 6.

Chapter 2

Related Work

As mentioned before, a complete IR-based QA system should contain three stages. However, in our case, since we have candidate choices, question processing techniques like answer type detection are unnecessary. So we focus on the latter two stages, table retrieval and answer selection. We first introduce several evaluation metrics in Section 2.1, and then review important works related to these two stages in the Section 2.2 and Section 2.3. There are not many works on multiple-choice question answering over semi-structured tables, so we consider it as a special case of multiple-choice question answering and review relative works in Section 2.3. Unless specified differently, we measure improvement with absolute improvement.

2.1 Evaluation Metrics

Table retrieval is a ranking problem, and the following IR metrics are commonly used to evaluate the performance of models.

Precision. Precision is the percentage of the retrieved tables that are relevant.

$$P(Q) = \frac{\#\text{ relevant tables that are retrieved}}{\#\text{ retrieved tables}} \tag{2.1}$$

where Q is a question.

Mean average precision@k (MAP@k). In order to define MAP@k, we need to define precision at position k (P@k) first.

$$P@k(Q) = \frac{\#\text{ relevant tables in top } k \text{ positions}}{k} \tag{2.2}$$

Average Precision@k (AP@k) is defined as:

$$AP@k(Q) = \frac{\sum_{n=1}^{k} P@n(Q) * l_n}{\text{\# relevant tables}} \tag{2.3}$$

where l_n is 1 if the table at position n is relevant, and 0 otherwise. Then MAP@k is the average of AP@k over all questions:

$$\text{MAP@k} = \frac{\sum_{Q=1}^{|\mathcal{Q}|} AP@k(Q)}{|\mathcal{Q}|} \tag{2.4}$$

where $|\mathcal{Q}|$ is the number of questions.

MAP@k is one of the most popular metrics in ranking tasks, and the choice of k depends on the task. If the number of tables to be ranked is small, or the task is very easy, people usually pay attention to the first few tables, and will choose small k like 1, 2 or 3. If the task is difficult, or the scale of the task is huge, people tend to use large k. If k is as large as the total number of tables, we can drop k and just denote it as MAP.

For answer selection, we can also evaluate the models with IR metrics like MAP@k. Another popular choice is accuracy, which is the percentage of correctly answered questions.

$$\text{Accuracy} = \frac{\text{\# correctly answered questions}}{\text{\# questions}} \tag{2.5}$$

Note, accuracy can be computed only when there's one correct result. And in this case, MAP@1 is equal to accuracy. But if there are more than one correct results, it's not appropriate to use accuracy to measure the performance of models.

2.2 Table Retrieval

Table retrieval is the task of retrieving tables whose cells are relevant to a question. In this section, we first review a special case of table retrieval, *ad hoc table retrieval*, where the question is a keyword or a phrase. Then we review existing works on table retrieval for natural language questions.

2.2.1 Ad Hoc Table Retrieval

Traditional ad hoc table retrieval methods are usually based on hand-engineered features, and focus on lexical similarity. Cafarella et al. [2] make one of the first efforts to tackle the task. They mainly compare two approaches. The first approach is to search the keyword in a search engine and extract tables from the top-ranked pages. The second approach is to train a linear regression model to estimate the similarities between the keyword and tables. They use a set of features to train the model, such as number of rows in the table, number of keywords occurrence in the header, etc. The results show that the trained estimator performs much better than the search-engine-based method. Pimplikar and Sarawagi [18] focus on keyword queries that describe the columns of tables, and frame the task and column mapping. They design a set of features to match the keywords against the header, content and context of the table to search for relevant columns. Then the tables with relevant columns are considered as relevant tables.

Recently, people have noticed that understanding the semantic of tables will help in ad hoc table retrieval. Zhang and Balog [41] design two ways to represent the semantic of keywords and tables: (1) *bag-of-concepts*: they use two discrete vectors to indicate the occurrence of entities and entity types, which are extracted with DBpedia; (2)*embeddings*: they map each word to a pretrained word embedding, and each entity to a pretrained graph embedding, and represent the keywords/tables as the average of word embeddings. Then they compute the cosine similarity between the representations of keywords and tables. And they add their semantic features to lexical features used in previous works, and get better performance than lexical features alone.

2.2.2 Table Retrieval for Question Answering

In order to retrieve tables for questions, understanding the semantic of both the question and the table is important. For example, *"What is Thanksgiving"* and *"When is Thanksgiving"* both ask about Thanksgiving, but the first asks about the definition or probably the history of Thanksgiving, and the second just asks about the date, and the answers to them could be in different tables. But an ad hoc table retrieval model may not be able to notice this difference.

Sun et al. [36] tackle the task with two steps. First, they retrieve a small set of candidate tables from a large collection of tables. In order to do this efficiently, they represent each question as bag-of-words, and represent each table as bag-of-words consists of its caption and headers. Then they compute the similarity score between question and table using BM25 [22]. Second, they build two models to rank the small set of tables: (1) *feature-based model*: they design word level features that capture word overlap, phrase level features that capture n-gram similarity, and sentence level features that capture semantic similarity between the question and header, content and caption of the table; (2) *neural model*: they use neural networks to compute the similarity between the question and the table caption, table headers, table content separately, based on semantic representations. More specifically, the question and caption of the table are encoded with a bi-directional gated recurrent units (BiGRU), and the headers and content of the table are represented with weighted average of word embeddings. Their results show that both feature-based model and neural model perform well, and the combination of both is even better.

Wang et al. [31] focus on ranking a small set of tables. Departing from previous methods, in their work, each row of a table is a complete sentence, and some of the columns do not have a name and only contain "link words" like "is". They concatenate the "link words" with the headers, making the headers also a meaningful sentence. For example, in the table in Figure 1.3, the headers concatenated with the link words will be "PHASE CHANGE — causes a — INITIAL PHASE — to change into a — FINAL PHASE — by — HEAT TRANSFER". Then they use neural networks like long short-term memory (LSTM) to convert the question, the caption of the table, and the headers of the table into dense vectors. They apply a fully-connected layer to integrate the vectors of table caption and headers to represent the table. Then they compute the cosine similarity between the representations of the question and the table.

2.3 Answer Selection

Answer selection is the task of identifying which of the candidate choices is the best answer. Here we review two related question answering tasks, *answer selection* and *multiple-choice reading comprehension*.

2.3.1 Answer Selection

Answer selection itself is actually an NLP task [14]. In this thesis, we need to select the correct answer from three or four choices based on some textual information, but in a more general sense, answer selection is the task of selecting the best answer from hundreds or even thousands of candidate choices with **no** context to refer to. In this case, the answer selection task is a *sentence pairing* problem [14].

The earliest works on sentence pairing rely on word overlap [11]. However, such approaches only learn the lexical part of sentences, and don't perform very well. So people start to utilize syntax information. Wang et al. [33] build a model that can match the dependency trees of sentence pairs. Later, Wang and Manning [32] proposed a CRF-based probabilistic model that models the alignment of sentence pairs as tree-edit operations on dependency trees. Heilman and Smith [7] use greedy search method to search for minimal edit sequences between dependency trees of sentence pairs, and extract 33 features from the edit sequences. Then they train a logistic regression classifier on the features. They get a 1.4% absolute increase of MAP over Wang and Manning's model on TRECQA dataset [33]. Yao et al. [37] further improve Heilman and Smith's approach by using dynamic programming to search for edit sequences. The MAP of their model is 2.8% higher than Heilman and Smith's.

Neural answer selection. Recently, many neural models have been proposed for this task. A typical neural answer selection model usually has two parts: a neural encoder that maps the sentences into dense vectors, and an interaction layer that measures the similarity (relevance) between the vectors [30]. Yu et al. [39] build two models to map the sentences into dense vectors: (1) *unigram model*: they use the average of word embeddings of all the words as the representation of a sentence; (2) *bigram model*: they use CNN to encode the sentence to capture bigram features. Their interaction layer is a fully-connected network. Their results show that the bigram model consistently outperforms the unigram model. That's because bag-of-words model is not aware of word order, which is important for sentence understanding. As both models are not able to capture string matching features, the authors add features that capture word co-occurrence, which improves the MAP by around 15%. And their method outperforms the non-neural method proposed by Yao et al. by 8%.

Tan et al. [28] propose QA-LSTM model for answer selection. They use a shared BiLSTM to encode the question and the answer, and then use a pooling layer to generate representations of the question and answer based on the hidden state vectors of the BiLSTM. Their interaction layer is simply cosine similarity. They further improve the model by adding an attention model to the answer encoder: before the pooling layer, the hidden state vector at each time step is multiplied by a weight, which is determined by the question representation. Their results show that the MAP of the system improves by about 2% with the attention model. Santos et al. [25] extend Tan et al.'s work by applying attention model to both question and answer encoder, and get around 4% improvement on accuracy. Instead of computing attention scores for only one sentence, Wang et al. [35] compute an attention score for every state vector of the two sentences. And their results show that computing the attention scores for both sentences results in 3% higher accuracy.

2.3.2 Multiple-Choice Reading Comprehension

In multiple-choice reading comprehension tasks, we need to choose the correct answer from a few, typically 4, choices based on a given document. This is similar to the answer selection task, but instead of mapping the question to an answer, we need to map the question and the answer to the document. Here we review several strategies to tackle this three-way matching problem.

The earliest work on multiple-choice reading comprehension by Richardson et al. [21] takes the intuitive way: they concatenate the question and the answer into a sentence, and match the sentence against the document. However, the concatenation of question and answer may not form a meaningful sentence. So, Sachan et al. [24] use a rule-based method to rewrite each question-choice pair into a meaningful statement.

Yin et al. [38] compare three strategies to tackle the task: (1) compute two document representations based on the question and one choice respectively, and then compare the two document representations; (2) compute the document representation based on the question, and compare it with the representations of choices; (3) same as Sachan et al. [24], they rephrase the question and one answer into a statement, and compare its representation with that of the document. Their results show that the third strategy works better than the other two by a margin ranging from 0.2% to 8.8%, depending on the experiment setting. However, there is yet no consensus on which strategy is the best. Lai et al. [13] and Zhu et al. [43] follow the second strategy, and Wang et al. [34] follow the first approach.

Multiple-choice question answering over tables. Multiple-choice question answering over tables is a task that hasn't received much attention. Jauhar [8] develops two models for multiple-choice question answering over tables. In his feature-based model, he design table-level, row-level, column-level and cell-level features to score question-answer pairs based on the tables. In his neural model, he maps each question to a row, the candidate choices to a column, and the intersection of the column and row to the answer. Wang et al. [31] use a neural model to compute the relevance score of every cell in the tables. Then the correct answer is extracted from the most relevant cells. The accuracy of Wang et al.'s method is 23.8% higher (relatively) than Jauhar's on TabMCQ dataset, and is the state-of-the-art method.

Chapter 3

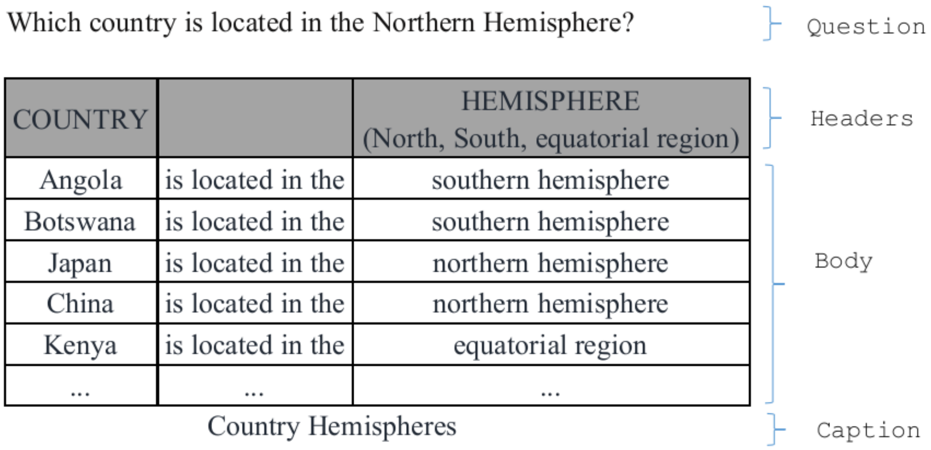
Table Retrieval

In this chapter, we study the table retrieval task. Following previous works [2], [18], [31], [36], [41], we tackle the task by scoring each table based on the question, and the table with the highest score is predicted as the relevant table.

Section 3.1 gives a formal definition of the table retrieval task. In Section 3.2, we present our feature-based model that focuses on lexical similarity. In Section 3.3, we introduce a neural model based on Wang et al.'s [31] work that focuses on semantic similarity.

3.1 Task Definition

We first introduce the notation needed to formulate the task. From this point on, we will use the terms question and query to mean the same thing: a natural language question. Let $Q = [q_1, q_2, ..., q_l]$ be a natural language query consisting of a list of tokens, where q_i is a token. Typically, a token can be a word, a number or punctuation. Let $\mathcal{T} = \{T_1, T_2..., T_N\}$ be a collection of tables, where T_i is a table. A table T consists of three fields: (1) *caption* T_c, which is a brief description of the table; (2) *headers* T_h, which is a list consisting of the names of columns; (3) *body* T_b, which is a list of table cells. Figure 3.1 gives an example of a query and a table, and illustrates different fields of a table. We can also divide the table collection $\mathcal{T}$ into three parts by the three fields: $\mathcal{T} = \{\mathcal{T}_c, \mathcal{T}_h, \mathcal{T}_b\}$. We use $\mathcal{T}_f$ to represent a field of the table collection ($f \in \{c, h, b\}$).



Which country is located in the Northern Hemisphere?

COUNTRY		HEMISPHERE (North, South, equatorial region)
Angola	is located in the	southern hemisphere
Botswana	is located in the	southern hemisphere
Japan	is located in the	northern hemisphere
China	is located in the	northern hemisphere
Kenya	is located in the	equatorial region
...	...	...

Country Hemispheres

Question
Headers
Body
Caption

Figure 3.1: An example of query-table pair.

The task of table retrieval is to automatically retrieve the most relevant table from a collection of tables. The input is a natural language query Q and a collection of tables $\mathcal{T}$, and the output is a table T from $\mathcal{T}$ that is most relevant to Q.

3.2 Feature-based Model

We design a feature-based model that learns to assign high score to the relevant table and low scores to irrelevant tables. The model takes a query Q and a table T as input, and extracts a feature vector from the query-table pair. Then the feature vector is fed into a fully-connected neural network to get a relevance score. The structure of the model is shown in Figure 3.2. Feature-based methods have been successfully used in table retrieval [2], [18], [36], [41].

Like Zhang and Balog [41], we divide our features into three categories: (1) query features; (2) table features; (3) query-table features. Table 3.1 summarizes the features we use. We adopt some features commonly used in document retrieval and table retrieval [19], [41], and design two novel features. We describe the details of the features we choose in the following sections.

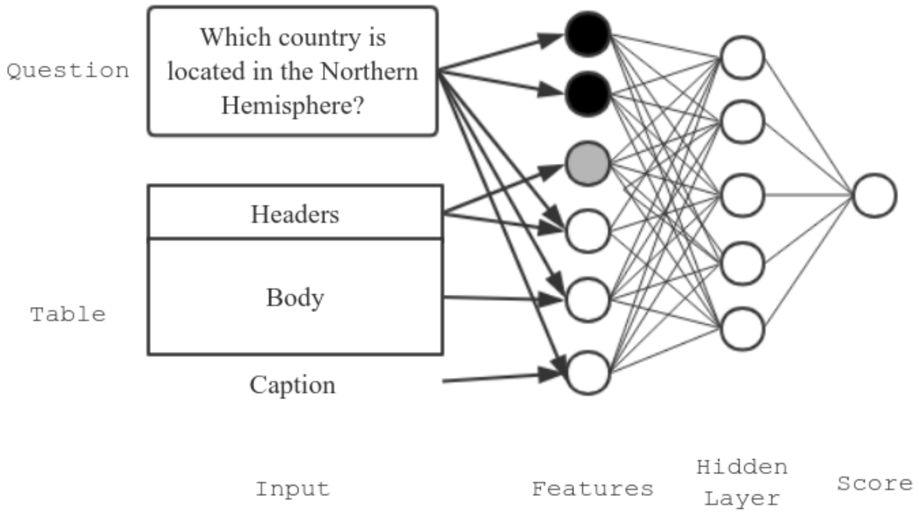


Figure 3.2: The structure of our feature-based model. Black circles represent query features, the grey circle represents table feature, and white circles represent query-table features. Details of the three kinds of features are given in Table 3.1.

3.2.1 Query Features

Query features are features that depend only on the query, and remain the same given different tables. We use one query feature, **QLEN**.

QLEN. QLEN is the number of tokens in the query.

3.2.2 Table Features

Table features are features that depend only on the table, and remain the same given any query. Many table features used in previous works are based on the webpage where the table is extracted, which are not available in our case. We only use one table feature, **#COL**.

#COL. #COL is the number of columns in the table.

3.2.3 Query-Table Features

Query-table features reflect the relationship between the query and table, and are dependent on both the query Q and the table T. We use five query-table features: **inverse document frequency (IDF)**, **term frequency(TF)**, **BM25**, **FUZZY** and **longest common substring (LCS) ratio**. To the

Query features	
QLEN	Number of tokens in the query
Table feature	
#COL	Number of columns in the table
Query-table features	
IDF	Sum, max and mean of query IDF scores in each field
TF	Sum, max and mean of query TF scores in each field
BM25	BM25 scores in each field
FUZZY	Sum, max and mean of query FUZZY scores in each field
LCS ratio	Normalized lengths of LCS of query and each field

Table 3.1: Features of our model. LCS stands for longest common substring. Novel features are marked with bold font.

best of our knowledge, FUZZY and LCS ratio are novel features that haven't been used in previous table retrieval methods.

1. **IDF**. IDF is a commonly used technique in information retrieval. The IDF score of a token q is the logarithmically scaled inverse fraction of the tables that contain q. We denote the number of tables that contain q as $n(q)$, and the total number of tables in $\mathcal{T}$ as N. IDF score of q is then computed as:

   $$IDF(q) = log\frac{N}{n(q)} \tag{3.1}$$

   IDF measures how much information a word provides. If a word occurs only in a small number of tables, it's likely to be more informative than frequent words like "the", and its IDF score will be high.

   Instead of treating the table collection $\mathcal{T}$ as a whole, Zhang and Balog [41] compute the IDF scores of tokens with respect to the three fields of $\mathcal{T}$. Thus, a token q has three IDF scores: $IDF_c(q)$, $IDF_h(q)$ and $IDF_b(q)$, each computed in the three fields of table. The IDF score in field f is computed as:

   $$IDF_f(q) = log\frac{N}{n_f(q)} \tag{3.2}$$

   where N is the number of tables, and $n_f(q)$ is the number of tables that contains q in field f. After computing the IDF scores for every token

$q \in Q \cap T_f$, we use the sum, max and mean of IDF scores in each field as features. Algorithm 1 shows the process of generating IDF features.

```
Algorithm 1: Generate IDF features
Data: A query $Q$, A table $T$
Result: An IDF feature vector $v_{IDF}$
$v_{IDF} \leftarrow emptylist$;
for each field $f \in \{c, h, b\}$ do
    $v_f \leftarrow emptylist$;
    for each token $q_i \in Q \cap T_f$ do
        Compute $IDF_f(q_i)$ according to Equation 3.2;
        Add $IDF_f(q_i)$ to $v_f$;
    end
    Add [sum($v_f$), max($v_f$), mean($v_f$)] to $v_{IDF}$;
end
return $v_{IDF}$
```

2. **TF**. The simplest form of term frequency is the count of the token in the table. We denote the count of token q in table T as $f_{q,T}$. However, raw count of the token favors long tables. So people often use the total count of tokens in T to normalize $f_{q,T}$. Term frequency of token q in table T is computed as:

$$tf(q,T) = \frac{f_{q,T}}{\sum_{q' \in T} f_{q',T}} \tag{3.3}$$

   Term frequency reflects the relevance between a term and a table. If $tf(q,T)$ is high, it's likely that q is relevant to T. We compute TF features in the same way as we compute IDF features. After computing the TF scores for every token in query Q in each field of table T, we use the sum, max and mean of the TF scores as features.

3. **BM25**. BM25 [22] is a ranking function commonly used in information retrieval. Different from IDF and TF, BM25 can score a query-table pair but not just a token-table pair. Given a table T, the BM25 score of query $Q = \{q_1, q_2, ..., q_l\}$ is:

$$BM25(Q,T) = \sum_{i=1}^{l} weight(q_i) \cdot relevance(q_i, T) \tag{3.4}$$

$$weight(q_i) = log\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \tag{3.5}$$

$$relevance(q_i, T) = \frac{f_{q_i,T} \cdot (k+1)}{f_{q_i,T} + k \cdot (1 - b + b \cdot \frac{|T|}{avgl})} \tag{3.6}$$

where $f_{q_i,T}$ is the count of token q_i in table T, N is the total number of tables, $n(q_i)$ is the number of tables that contain q_i, $|T|$ is the length (number of tokens) of table T, and avgl is the average length of tables in $\mathcal{T}$. k and b are hyperparameters to be tuned.

BM25 can be viewed as a weighted sum of relevance scores between tokens in the query and the table. Equation 3.5 measures the importance of token q_i, and Equation 3.6 measures the relevance between q_i and table T. BM25 score will be high if Q and T share rare words. We compute BM25 scores of Q with respect to the three fields of the tables.

4. **FUZZY**. Typos are common in queries and tables, and can be vital to relate a table to a query. If the query contains "protin", it's likely that tables contain "protein" are relevant to the query. However, token level features like IDF, TF and BM25 are based on exact match, and cannot make use of such information. So we design a novel feature FUZZY, which can catch character level information and fuzzily match tokens. We use the following equation to compute the character level similarity of two tokens q_i and q_j:

$$sim(q_i, q_j) = 1 - \frac{Ldist(q_i, q_j)}{|q_i| + |q_j|} \tag{3.7}$$

where $Ldist(q_i, q_j)$ is Levenshtein distance between q_i and q_j, and $|q_i|$ is the length (number of characters) of q_i. Levenshtein distance is widely used to measure the similarity between strings. It is the minimum number of single-character edits (insertions, deletions or substitutions) that is required to convert a string into another. The time complexity to compute Levenshtein distance is $O(|q_i| * |q_j|)$. Intuitively, the smaller the Levenshtein distance is, the more similar the two strings are. Since Levenshtein distance tends to be larger for longer strings, we normalize it with the lengths of strings.

In order to score table T given token q, we just search for the token that is most similar to q.

$$FUZZY(q,T) = \max_{t_i \in T} sim(q,t_i) \tag{3.8}$$

Since we aim at utilizing information in possible typos, we only compute FUZZY scores for tokens that are not in the vocabulary of tables. Algorithm 2 shows the process of generating FUZZY features.

```
Algorithm 2: Generate FUZZY features
  Data: A query $Q$, a table $T$, the table vocabulary $vocab$
  Result: A FUZZY feature vector $v_{FUZZY}$
$v_{unk} \leftarrow emptylist$;
for each token $q_i \in Q$ do
    if $q_i$ not in vocab then
        Add $q_i$ to $v_{unk}$;
    end
end
if $v_{unk}$ is not empty then
    $v_{FUZZY} \leftarrow emptylist$;
    for each field $f \in \{c, h, b\}$ do
        $v_f \leftarrow emptylist$;
        for each token $v_i \in v_{unk}$ do
            Compute $FUZZY(v_i, T_f)$ according to Equation 3.8;
            Add $FUZZY(v_i, T_f)$ to $v_f$;
        end
        Add [sum($v_f$), max($v_f$), mean($v_f$)] to $v_{FUZZY}$;
    end
else
    $v_{FUZZY} = [0,0,0,0,0,0,0,0,0]$
end
return $v_{FUZZY}$
```

5. **LCS ratio**. The features described above all treat the query Q and table T as bags of tokens and are not sensitive to word order. We design a new feature "LCS ratio" to capture word order information. Longest common substring is the longest string that is a substring of two strings. The longer the LCS of Q and T is, the more likely that they are relevant. Because LCS favors long queries, we normalize it with the length of the

query. LCS ratio is computed as:

$$LCSratio(Q,T) = \frac{|LCS(Q,T)|}{|Q|} \tag{3.9}$$

where $|LCS(Q,T)|$ is the length (number of characters) of LCS of Q and T, and $|Q|$ is the length of Q. We compute LCS ratio with respect to the three fields of tables individually.

3.2.4 Table scoring

After getting all the features, we concatenate the features into a vector $\mathbf{v} \in \mathbb{R}^d$, and feed the feature vector $\mathbf{v}$ into a fully-connected neural network with one hidden layer. The relevance score is computed as:

$$score(Q,T) = sigmoid(V^\top tanh(W\mathbf{v} + \mathbf{b_1}) + b_2) \tag{3.10}$$

where $W \in \mathbb{R}^{d' \times d}$ is a trainable matrix, $V \in \mathbb{R}^{d'}$ is a trainable vector, $\mathbf{b_1}$ and b_2 are bias units, d' is the size of the hidden layer.

3.2.5 Training

The ground truth score for related query-table pair is 1. Besides, for each query, we randomly select 2 unrelated tables, and set the ground truth score to 0. We use binary cross entropy as loss function. The model is trained with Adadelta [40] optimizer.

3.3 Neural Network Model

Neural network haven't been widely used in table retrieval tasks. One possible reason is that it's difficult to convert the table, especially the table body into a meaningful vector representation. We follow the methodology of Wang et al. [31], and convert the caption, headers and part of the body into a vector to represent the table. After converting the query and table into vectors, we measure the relevance between the query and the table based on cosine similarity. The architecture of the neural model is given in Figure 3.3.

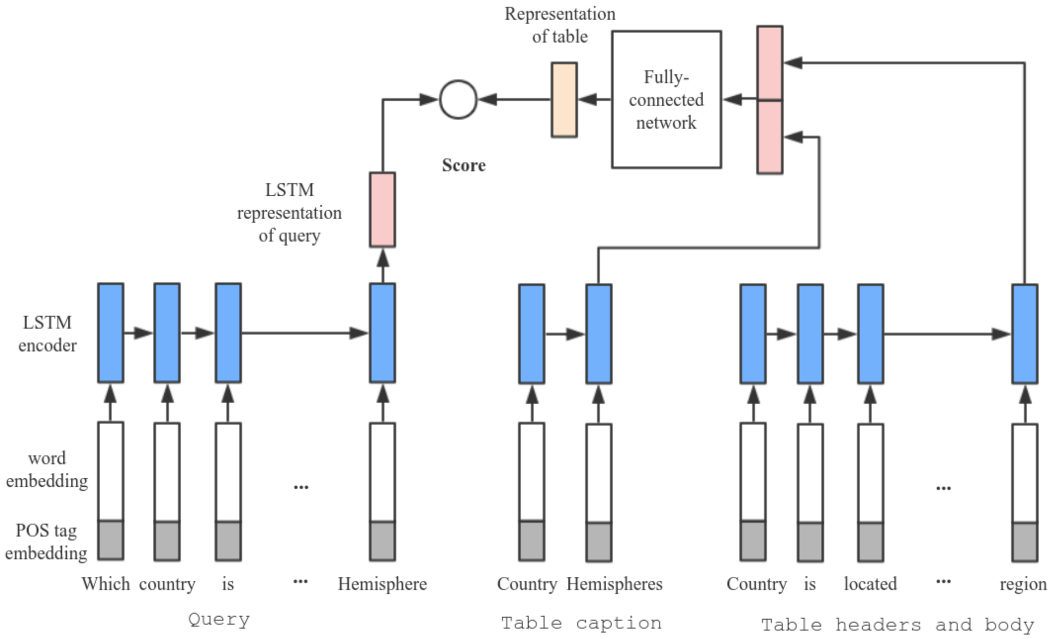
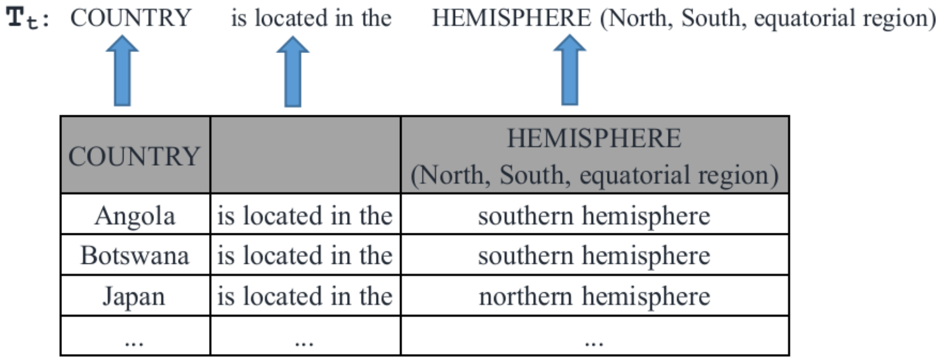


Figure 3.3: The architecture of neural model.

3.3.1 Input Representation

The input to the model is a query $Q = \{q_1, q_2, ..., q_l\}$ and a table $T = \{T_c, T_h, T_b\}$. T_c is typically a meaningful phrase or sentence consists of several tokens $\{t_{c_1}, t_{c_2}, ..., t_{c_m}\}$. Some columns do not have a header, like the second column of the table in Figure 3.1. Those columns contain "link words" that can complete a sentence, like "is located in". We concatenate the headers and link words into a word sequence $T_t = \{t_{t_1}, t_{t_2}, ..., t_{t_n}\}$ to represent the headers and body of the table. Figure 3.4 illustrates an example of converting the table headers and body to T_t. So a table is represented with $T = \{T_c, T_t\}$.

We represent each token with a dense vector. Given a word embedding matrix $W^{emb} \in \mathbb{R}^{d_w \times |V|}$, we can map a token w_i to a column vector $\mathbf{w}_i^d \in \mathbb{R}^{d_w}$, where d_w is the dimension of word embedding, and $|V|$ is the size of vocabulary. To capture the information of part-of-speech (POS) tags, we concatenate POS embeddings to the word embedding. We map every POS tag to a vector of size d_p. So the final representation of token w_i is $\mathbf{w}_i^E = [\mathbf{w}_i^d \oplus \mathbf{w}_i^p]$, where $\mathbf{w}_i^p$ is the POS embedding for token w_i, and $\oplus$ denotes concatenation operation.


$\mathbf{T_t}$: COUNTRY is located in the HEMISPHERE (North, South, equatorial region)

COUNTRY		HEMISPHERE (North, South, equatorial region)
Angola	is located in the	southern hemisphere
Botswana	is located in the	southern hemisphere
Japan	is located in the	northern hemisphere
...	...	...

Figure 3.4: An example of joint representation of table headers and table body.

3.3.2 Query Representation

We denote the representation of the query Q as $E_Q = [\mathbf{w}^E_{q1}, \mathbf{w}^E_{q2}, ..., \mathbf{w}^E_{ql}]$, where $E_Q \in \mathbb{R}^{(d_w+d_p)\times l}$. To convert E_Q into a vector that contains semantic information, we apply a standard LSTM to E_Q from left to right, and the final output $h_Q \in \mathbb{R}^{d_l}$ is considered as the LSTM representation of Q, where d_l is the state size of the LSTM.

$$h_Q = LSTM(E_Q) \tag{3.11}$$

3.3.3 Table Representation

We denote the representation of T_c and T_t as $E_{T_c} \in \mathbb{R}^{(d_w+d_p)\times m}$ and $E_{T_t} \in \mathbb{R}^{(d_w+d_p)\times n}$ respectively, both consist of the vector representations of the tokens in them. We first convert E_{T_c} and E_{T_t} into a vector of size d_l using a standard LSTM:

$$h_{T_c} = LSTM(E_{T_c}) \tag{3.12}$$

$$h_{T_t} = LSTM(E_{T_t}) \tag{3.13}$$

Then we use a transformation matrix to transform h_{T_c} and h_{T_t} into the vector representation of the whole table:

$$h_T = W[h_{T_c} \oplus h_{T_t}] + \mathbf{b} \tag{3.14}$$

where $W \in \mathbb{R}^{d_l\times 2d_l}$ is the trainable transformation matrix, and $\mathbf{b}$ is a bias vector.

3.3.4 Table Scoring

The cosine similarity between h_Q and h_T is computed as:

$$cos(Q,T) = \frac{h_Q \cdot h_T}{|h_Q||h_T|} \tag{3.15}$$

where $|h_Q|$ denotes the magnitude of vector h_Q.

Then we apply sigmoid function to $cos(Q,T)$ to get the relevance score of Q and T:

$$rel(Q,T) = sigmoid(k \cdot cos(Q,T) + b) \tag{3.16}$$

where k is a parameter to be learned during training, b is a bias unit.

3.3.5 Training

As in feature-based model, the ground truth score for related query-table pair is 1. Besides, for each query, we randomly select 2 unrelated tables, and set the ground truth score to 0. We use binary cross entropy as loss function. The model is trained with Adadelta [40] optimizer.

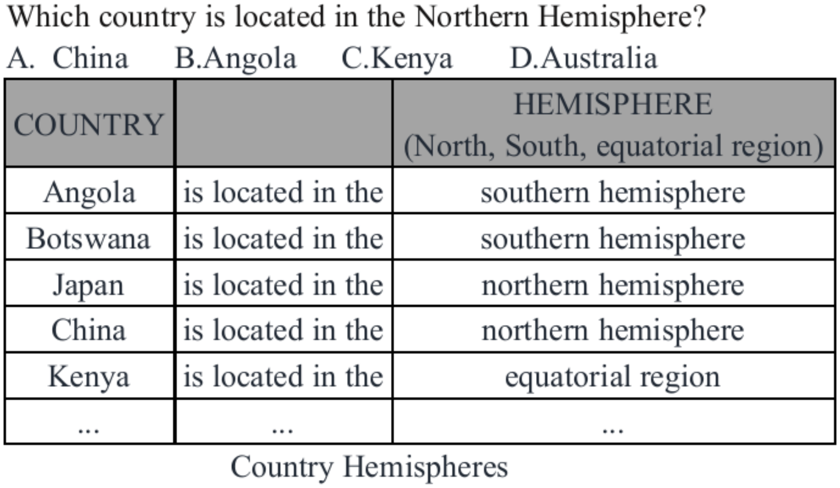
3.4 Summary

In this chapter, we introduce a feature-based model and a neural network model to tackle the task of table retrieval. The feature-based model mainly focuses on lexical similarity. Previous works often use features that captures only exact match of tokens, but we add two novel features that can capture character level information and word order information. Neural model is less explored in this task, and we follow the methodology of Wang et al. [31]. We jointly represent the headers and body of the table with “link words” and headers, and encode the query and table into vectors with LSTM to capture the semantic information. And the relevance score is computed based on the vector representations of query and table.

Chapter 4

Answer Selection

After retrieving the relevant table for a query, we need to select the best answer among the choices. Jauhar [8] tackles the problem in three steps: (1) map the query to a row r; (2) map the candidate choices to a column c; (3) map the cell $T(r, c)$ to a choice. However, this method doesn't work very well because of the first step. Let's look at the example in Figure 4.1. It is very difficult to map the query "*Which country is in the Northern Hemisphere?*" to a certain row even by human, because the rows like "Japan | is located in the | northern hemisphere" and "China | is located in the | northern hemisphere" are equally relevant to the query. If we map the query to "Japan | is located in the | northern hemisphere", we will not be able to correctly select the answer based on it.



Which country is located in the Northern Hemisphere?
A. China B.Angola C.Kenya D.Australia

COUNTRY		HEMISPHERE (North, South, equatorial region)
Angola	is located in the	southern hemisphere
Botswana	is located in the	southern hemisphere
Japan	is located in the	northern hemisphere
China	is located in the	northern hemisphere
Kenya	is located in the	equatorial region
...	...	...

Country Hemispheres

Figure 4.1: An example of answer selection.

Instead of mapping the query Q to a row r, Wang et al. [31] directly score each query-cell pair. And in order to make sure that the cells in the same context get the same score, like "Japan" and "China" in the above example, they replace the cell to be scored with a special token $< SPACE >$. Then the row "$< SPACE >$ | is located in the | northern hemisphere" is called a "pattern". Then they score the query-cell pair by comparing the query with the pattern.

We tackle the task in three steps: pattern extraction, pattern scoring and answer selection. Our approach is similar to Wang et al.'s, but with three major improvements. First, instead of scoring each cell in the table, we only score cells in a certain column. Second, we augment the neural model with additional features. Third, we develop the answer selection algorithm which is essential to the system but is not explicitly described in Wang et al.'s paper.

Section 4.1 gives the definition of answer selection task. Then we introduce our method to tackle the task in Section 4.2, Section 4.3 and Section 4.4.

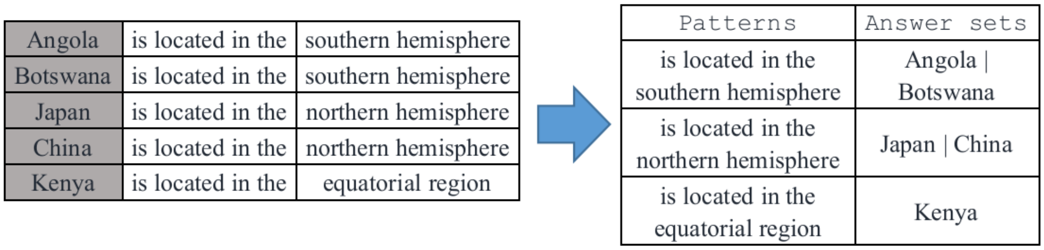
4.1 Task Definition

We use $Q = [q_1, q_2, ..., q_l]$ to denote a query, $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ to denote the candidate choices for the query, where C_i is a choice consisting of several tokens. We denote a table T only with the cells in the body, $T = \{T(j,k) | j \in [1, d_r^T], k \in [1, d_c^T]\}$, where $T(j,k)$ is the cell at row j and column k, d_r^T and d_c^T are the number of rows and the number of columns of table T respectively.

Given a query Q with several candidate choices $\mathcal{C} = \{C_1, C_2, ..., C_n\}$, the task of answer selection is to select the best answer C from $\mathcal{C}$ based on a table T.

4.2 Pattern Extraction

As mentioned before, Wang et al. use neural networks to score each query-cell pair. However, we can filter out some irrelevant cells quickly before sending them to the neural networks. Usually, candidate choices describe similar things. For example, the candidate choices for the query "*Which country is in*



Angola	is located in the	southern hemisphere
Botswana	is located in the	southern hemisphere
Japan	is located in the	northern hemisphere
China	is located in the	northern hemisphere
Kenya	is located in the	equatorial region

Patterns	Answer sets
is located in the southern hemisphere	Angola | Botswana
is located in the northern hemisphere	Japan | China
is located in the equatorial region	Kenya

Figure 4.2: Illustration of pattern extraction. The table on the left is the original table, and the column in grey is the relevant column. The table on the right illustrates the extracted patterns and the corresponding answer sets.

the Northern Hemisphere?" are likely to be countries, and they only relate to a single column. Thus, we can map the candidate choices to a single column, and only score the cells in that column. And to avoid repeatedly scoring the cells in the same pattern, we extract patterns after removing the cells, and score the query-pattern pair instead of query-cell pair. Figure 4.2 gives an example of pattern extraction.

4.2.1 Column Selection

We select the relevant column by string matching. Here we use Jaccard n-gram to measure string similarity. The Jaccard similarity between two sets A and B is the size of the intersection of A and B divided by the size of the union of A and B:

$$JS(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

To compute the Jaccard n-gram similarity between strings s_i and s_j, we first turn them into sets of n-grams. An n-gram of a string is a contiguous substring of n characters from it. For example, "gre" and "rea" are 3-grams of "great". Then we compute the Jaccard similarity using Equation 4.1.

For each column, we first compute its relevance score with one choice. To do so, we go through the cells in the column and compute the Jaccard n-gram similarity of each cell-choice pair, and use the highest score as the score of the column-choice pair. We compute its relevance score with each choice, and use the sum of the scores to measure the relevance of the column. And the column with the highest score is selected as the relevant column. Algorithm 3 shows the process of column selection.

Algorithm 3: Column selection

```
Data: A table $T$, candidate choice set $\mathcal{C}$
Result: The most relevant column $c$
$c \leftarrow None$;
$maxscore \leftarrow -1$;
for each column $k \in T$ do
    $rel(k) \leftarrow 0$;
    for each candidate choice $C \in \mathcal{C}$ do
        $score \leftarrow 0$;
        for each cell $T(j,k) \in T(\cdot,k)$ do
            Compute $JS(C, T(j,k))$ according to Equation 4.1;
            $score \leftarrow \max(score, JS(C, T(j,k)))$;
        end
        $rel(k) \leftarrow rel(k) + score$;
    end
    if $rel(k) > maxscore$ then
        $maxscore \leftarrow rel(k)$;
        $c \leftarrow k$;
    end
end
return $c$
```

4.2.2 Pattern Extraction

After finding the most relevant column, we just remove the cells in that column, and merge the cells in the same pattern into an answer set. In this way, we can make sure that the cells in the same context are in the same answer set, like “Japan” and “China” in Figure 4.2.

4.3 Pattern Scoring

We use a neural model to compute the similarity score between the query and the patterns. We first apply a soft attention layer to make the model focus on useful information of the patterns. Then we use bidirectional LSTM (BiLSTM) to encode the query and the pattern separately into dense vectors. Then we feed the concatenation of the encodings of the query and the pattern, together

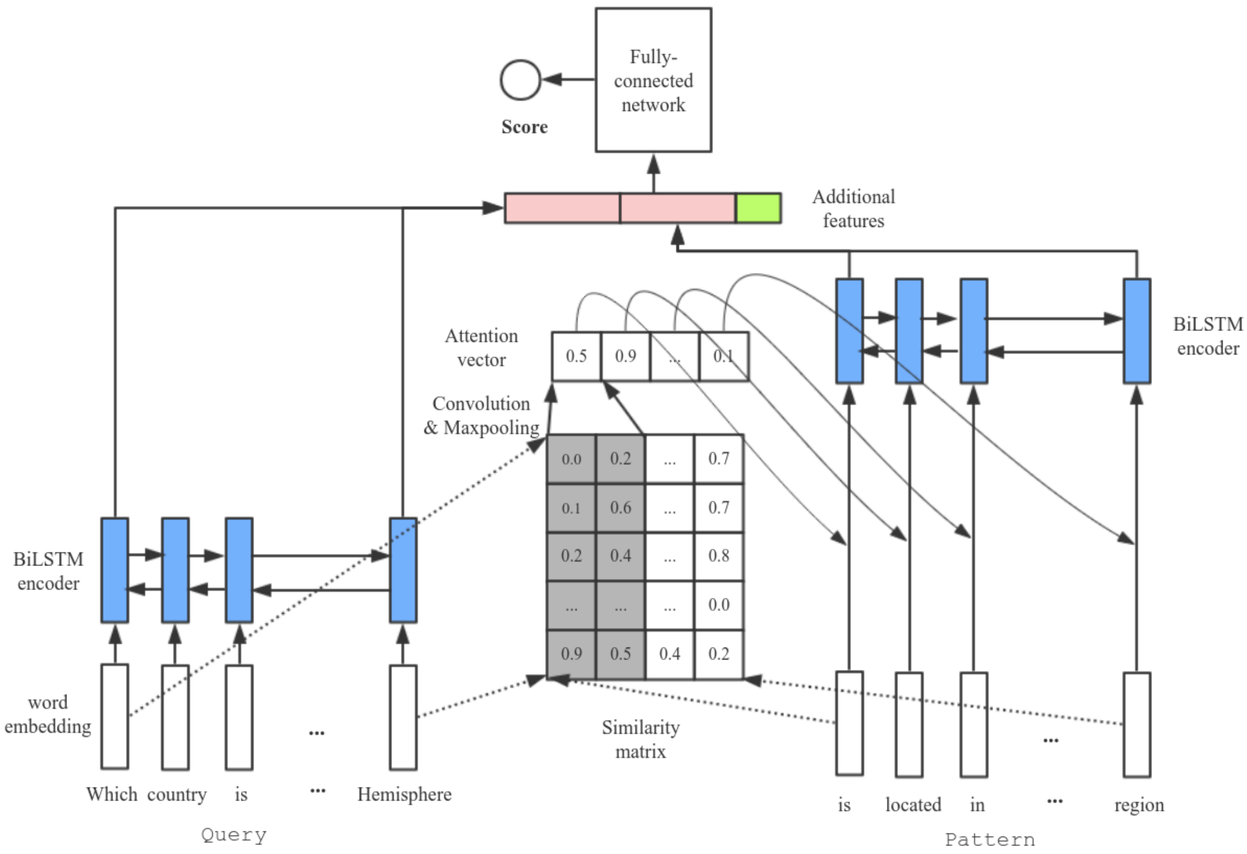


Figure 4.3: Structure of the pattern scoring model.

with additional features to a fully-connected network to get the similarity score. The structure of the model is given in Figure 4.3.

4.3.1 Input Representation

The input to the model is a query $Q = \{q_1, q_2, ..., q_l\}$ and a pattern $P = \{p_1, p_2, ..., p_m\}$, each consists of a sequence of tokens. We represent each token with a dense vector. Given a word embedding matrix $W \in \mathbb{R}^{d_w \times |V|}$, we can map a token w_i to a column vector $\mathbf{w}_i \in \mathbb{R}^{d_w}$, where d_w is the dimension of word embedding, and $|V|$ is the size of vocabulary.

4.3.2 Query Representation

We first represent query Q with a matrix consisting of the vector representations of the tokens, which is denoted as $E_Q = [\mathbf{w}^E_{q_1}, \mathbf{w}^E_{q_2}, ..., \mathbf{w}^E_{q_l}]$, where

$E_Q \in \mathbb{R}^{d_w \times l}$. To convert E_Q into a vector that contains semantic information, we apply BiLSTM to E_Q, and the final states of two directions of BiLSTM are concatenated into a vector h_Q as the representation of Q:

$$h_Q = BiLSTM(E_Q) \tag{4.2}$$

where $h_Q \in \mathbb{R}^{2d_l}$, and d_l is the state size of LSTM.

4.3.3 Pattern Representation

To help the model to focus on important parts, we apply a soft attention layer like Wang et al.. We first compute a similarity matrix $S_{Q,P} = \{cos(\mathbf{w}_{q_i}, \mathbf{w}_{p_j})|i \in [1, l], j \in [1, m]\}$, where $cos(\mathbf{w}_{q_i}, \mathbf{w}_{p_j})$ is the cosine similarity between the vector representations of q_i and p_j:

$$cos(\mathbf{w}_{q_i}, \mathbf{w}_{p_j}) = \frac{\mathbf{w}_{q_i} \cdot \mathbf{w}_{p_j}}{|\mathbf{w}_{q_i}||\mathbf{w}_{p_j}|} \tag{4.3}$$

where $|\mathbf{w}|$ is the magnitude of vector $\mathbf{w}$. Then we feed $S_{Q,P}$ to a convolution layer followed by a max-pooling layer. And we apply a *tanh* layer to the output of pooling layer to get the attention weight vector $\mathbf{a} \in \mathbb{R}^m$:

$$\mathbf{a} = tanh(pool(conv(S_{q,p}))) \tag{4.4}$$

The attention weight for token p_i is $\mathbf{a}_i$. And to represent token p_i, we multiply its vector representation $\mathbf{w}_{p_i}$ with $\mathbf{a}_i$. We can represent the pattern P with a matrix $E_P = [\mathbf{a}_1\mathbf{w}_{p_1}, \mathbf{a}_2\mathbf{w}_{p_2}, ..., \mathbf{a}_m\mathbf{w}_{p_m}]$, where $E_P \in \mathbb{R}^{d_w \times m}$.

Like in query representation, we also use a BiLSTM to convert E_P into a vector $h_P \in \mathbb{R}^{2d_l}$:

$$h_P = BiLSTM(E_P) \tag{4.5}$$

4.3.4 Pattern Scoring

h_Q and h_P capture the semantic information of query Q and pattern P. Wang et al. [31] then feed $[h_Q \oplus h_P]$ to a fully-connected network with one hidden layer to get the similarity score of Q and P. Inspired by previous works on text matching [26], [29], we add more features to augment the model.

As noted by Yu et al. [39], models based on distributed representation of words are not sensitive to numbers and proper nouns. So we add a feature vector X_{feat} that captures lexical similarity between Q and P. X_{feat} contains two features: count of same tokens in Q and P, count of same non-stop words in Q and P.

We also compute the bilinear similarity between h_Q and h_P:

$$sim(Q, P) = h_Q^\top M h_P \tag{4.6}$$

where $M \in \mathbb{R}^{2d_l \times 2d_l}$ is the similarity matrix, which is learned during training.

Then the input to the fully-connected network is $X_{join} = [h_Q \oplus h_P \oplus sim(Q, P) \oplus X_{feat}]$. And the final score is computed as:

$$score(Q, P) = sigmoid(V^\top tanh(WX_{join} + \mathbf{b}_1) + b_2) \tag{4.7}$$

where $W \in \mathbb{R}^{d' \times (4d_l+3)}$ is a trainable matrix and $V \in \mathbb{R}^{d'}$ is a trainable vector, d' is the size of the hidden layer, $\mathbf{b}_1$ and b_2 are bias units.

4.3.5 Training

The ground truth score for related query-pattern pair is set to 1. Besides, for each query, we randomly select 2 unrelated patterns, and set the ground truth score to 0. We use binary cross entropy as loss function. The model is trained with Adadelta [40] optimizer.

4.4 Answer Selection

After scoring each pattern, we need to select the best answer from the candidate choices $\mathcal{C}$.

We first order all answer sets by the scores of corresponding patterns, and then match the cells in the first answer set with the candidate choices. Exact match of cell-choice doesn't work well because the same term may be expressed in different ways in the choices and the cells, like “monkey” and “monkeys”. So we use FUZZY similarity (defined in Equation 3.7) to match the cell-choice pairs. If we cannot find a satisfying cell-choice pair in the first answer set, we move to the next, until we find a satisfying cell-choice pair. In order to measure if the cell-choice pair is satisfying, we set a threshold θ, and if the FUZZY similarity of the cell-choice pair is greater than θ, we decide that the correct answer is found. θ is a hyperparameter that needs to be tuned.

However, if our column selection is wrong, or θ is too high, it's possible that we can never find a satisfying cell-choice pair. In this case, we will just select the correct answer based on the first answer set, and the choice that matches best with one of the cells is predicted as the answer. The procedure of answer selection is given in Algorithm 4.

4.5 Summary

In this chapter, we introduced our answer selection model. We improve the previous work in several aspects. First, we filter out irrelevant cells by column selection. Second, we add additional features to the neural model to capture lexical similarity. Third, we develop an effective answer selection algorithm.

Algorithm 4: Answer selection

```
**Data:** Candidate choice set $\mathcal{C}$, ordered answer sets $\mathcal{A}$, threshold $\theta$
**Result:** The answer $Correct$
$Correct \leftarrow None$;
**for** *each answer set* $A \in \mathcal{A}$ **do**
    $maxscore \leftarrow -1$;
    $bestchoice \leftarrow None$;
    **for** *each candidate choice* $C \in \mathcal{C}$ **do**
        Compute $FUZZY(C, A)$ according to Equation 3.8;
        **if** $FUZZY(C, A) > maxscore$ **then**
            $maxscore \leftarrow FUZZY(C, A)$;
            $bestchoice \leftarrow C$;
        **end**
    **end**
    **if** $maxscore > \theta$ **then**
        $Correct \leftarrow bestchoice$;
        Terminate for loop;
    **end**
**end**
**if** *Correct is None* **then**
    $A \leftarrow$ first answer set in $\mathcal{A}$;
    $maxscore \leftarrow -1$;
    **for** *each candidate choice* $C \in \mathcal{C}$ **do**
        Compute $FUZZY(C, A)$ according to Equation 3.8;
        **if** $FUZZY(C, A) > maxscore$ **then**
            $maxscore \leftarrow FUZZY(C, A)$;
            $Correct \leftarrow C$;
        **end**
    **end**
**end**
**return** $Correct$
```

Chapter 5

Experiments

In this chapter, we report on experiments to evaluate the performance of our models. Section 5.1 introduces the dataset we use. We evaluate and analyze our table retrieval models in Section 5.2, and then evaluate and analyze our answer selection model in Section 5.3.

5.1 Dataset

We evaluate the proposed models on a publicly available dataset, TabMCQ [1]. TabMCQ is a manually annotated multiple-choice question answering dataset. It contains 9092 *4th grade science exam* queries, each with three or four candidate choices (mostly four). The dataset has 63 tables, and each query is related to one table. The tables come from AI2's Aristo Tablestore. In the dataset, each example contains the following information:

1. QUERY: the actual text of the query.

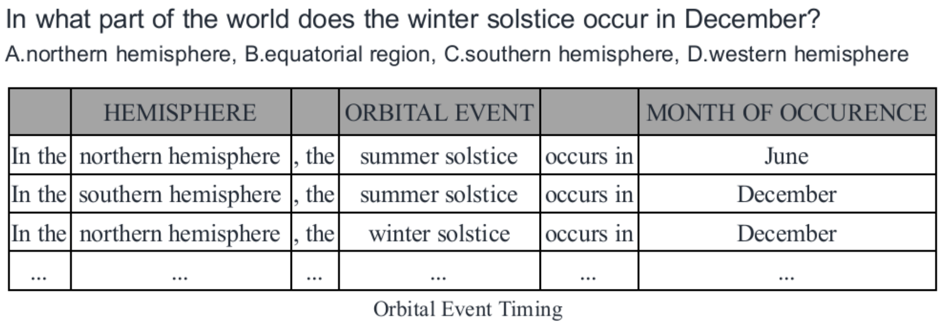
2. CANDIDATE CHOICES: the different choices for the query.

3. CORRECT CHOICE: the correct answer to the query.

4. RELEVANT TABLE: the table needed to answer this query.

5. RELEVANT ROW: the row in the RELEVANT TABLE needed to answer the query.

[1] http://ai2-website.s3.amazonaws.com/data/TabMCQ_v_1.0.zip



In what part of the world does the winter solstice occur in December?

A.northern hemisphere, B.equatorial region, C.southern hemisphere, D.western hemisphere

	HEMISPHERE		ORBITAL EVENT		MONTH OF OCCURENCE
In the	northern hemisphere	, the	summer solstice	occurs in	June
In the	southern hemisphere	, the	summer solstice	occurs in	December
In the	northern hemisphere	, the	winter solstice	occurs in	December
...	...	...	...	...	...

Orbital Event Timing

Figure 5.1: An example from TabMCQ.

6. RELEVANT COL: the column in the RELEVANT ROW of the RELEVANT TABLE containing the answer to the query.

Figure 5.1 shows an MCQ and its corresponding table from the dataset. Table 5.1 shows statistics of the dataset.

Number of tables	63
Number of queries	9092
Queries per table	144.3
Avg number of rows	61.5
Max number of rows	1217
Min number of rows	2
Avg number of cols	4.3
Max number of cols	9
Min number of cols	2

Table 5.1: Statistics of TabMCQ dataset.

5.2 Table Retrieval Experiments

5.2.1 Experiment Setup

When preprocessing the dataset, we first remove all punctuation, and then use the tokenizer from NLTK [1] to split the queries and tables into lists of tokens. All letters are set to lower case. To calculate the LCS ratio features (defined in Section 3.2.3), we concatenate the tokens in queries and tables into strings.

In the neural model, the vector representations of words are initialized with GloVe word embedding by Pennington et al. [16], while the POS tag embeddings are randomly initialized. Both word embeddings and POS tag embeddings are fine-tuned during training.

Baseline: We use BM25 as baseline. The explanation of BM25 can be found in Section 3.2.3. To compute the BM25 score of a table given a query, we treat the whole table (caption, headers and body) and the query as lists of words, and then compute the score of the table using the BM25 function.

Evaluation Metrics: Following previous works [31], [36], we use Mean Average Precision@k (MAP@k) to evaluate our model. MAP@k is defined in Section 2.1.

Hyperparameters: We use 10% of queries in training set as validation set to tune hyperparameters for the feature-based model and neural model. We first set the hyperparameters to default value or commonly used value, and then search for the optimal hyperparameters one by one. In the neural model, we apply a dropout layer to both input and output of every LSTM layer. The hyperparameters are given in Table 5.2.

Hyperparameters for feature-based model	
k in BM25	1.3
b in BM25	0.75
size of hidden layer	32
Hyperparameters for neural model	
state size of LSTM	64
dropout	0.2
POS tag embedding dimension	16

Table 5.2: Hyperparameters for feature-based and neural models.

To reduce the noise introduced by randomly generating query-table pairs for training, we set the random seed to 0. We run the models using the tuned hyperparameters for 5 times and report the **average** MAP@k.

5.2.2 Performance Comparison

Previous works on table retrieval only compares queries with tables. However, in TabMCQ, candidate choices can offer additional information for table retrieval. We can concatenate the candidate choices to queries to utilize this information. We report the results with 95% confidence intervals under both settings (with/without candidate choices) in Table 5.3. And it's also possible to ensemble the feature-based model and the neural model. To do so, for each query, we first normalize the scores of tables, and sum up the scores given by the two models as the final scores.

Model	MAP@1	MAP@2	MAP@3
BM25	89.2%	91.3%	92.0%
Feature	95.4% ± 0.3%	96.5% ± 0.1%	96.8% ± 0.1%
Neural	95.1% ± 0.6%	96.5% ± 0.4%	96.8% ± 0.3%
Feature + Neural	**97.3% ± 0.2%**	**98.2% ± 0.1%**	**98.4% ± 0.1**
BM25 w/cc	95.9%	97.0%	97.2%
Feature w/cc	**99.7% ± 0.1%**	99.8% ± 0.0%	99.8% ± 0.0%
Neural w/cc	96.5% ± 0.2%	97.7% ± 0.2%	98.0% ± 0.2%
Feature + Neural w/cc	**99.8% ± 0.0%**	**99.9% ± 0.0%**	**99.9% ± 0.0%**

Table 5.3: MAP@k for BM25 and our models. w/cc stands for "with candidate choices". Highest scores under each setting are marked with bold font.

From the result, we can see that when we do not use candidate choices, the feature-based model performs comparably well with the neural model, and they both outperform the BM25 baseline. This shows the effectiveness of using lexical similarity. However, both BM25 and the feature-based model perform much better if we use candidate choices, but the improvement of the neural model is not very large. As a result, the neural model can only perform comparably with the BM25 baseline. The reason could be that the candidate choices are typically keywords that can be found in the body of the table, but the neural model cannot effectively utilize the table body. So the neural model doesn't benefit much from the additional information provided by the candidate choices. The ensemble model is better than feature-based or neural model alone in both settings, which implies the complementation between them.

5.2.3 Ablation Study of Feature-based Model

To evaluate the contribution of our novel features, FUZZY and LCS ratio, we do ablation study, by removing one novel feature from the model. The results are shown in Table 5.4.

Model	MAP@1	MAP@2	MAP@3
Full model	95.4% ± 0.3%	96.5% ± 0.1%	96.8% ± 0.1%
-FUZZY	94.7% ± 0.3%	96.1% ± 0.2%	96.3% ± 0.2
-LCS ratio	94.6% ± 0.4%	96.0% ± 0.3%	96.3% ± 0.2%
Full model w/cc	99.7% ± 0.1%	99.8% ± 0.0%	99.8% ± 0.0%
-FUZZY w/ cc	99.3% ± 0.2%	99.7% ± 0.1%	99.7% ± 0.1%
-LCS ratio w/cc	99.6% ± 0.1%	99.8% ± 0.0%	99.8% ± 0.0%

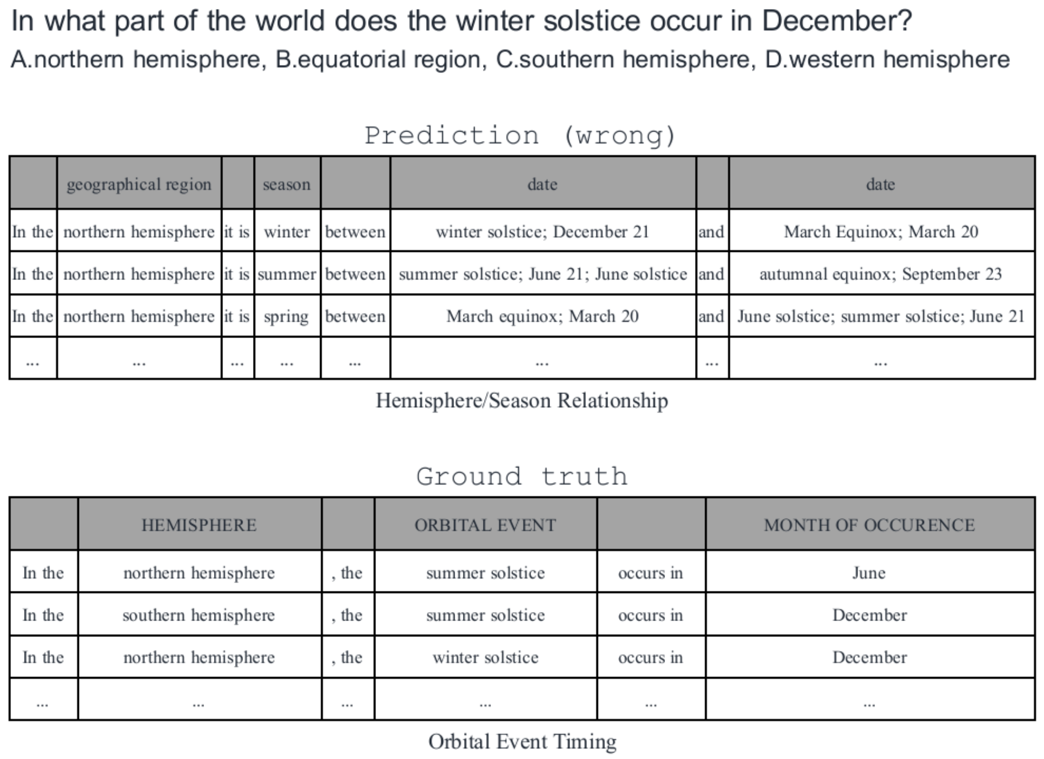
Table 5.4: Ablation study of novel features.

From the results, we can see that FUZZY can bring significant improvement under both settings. FUZZY meets our expectation and can find typos like "kilgrams", and correctly relate the typos to tables that contain the correct forms. What's more, we find that FUZZY can even reveal some semantic information. For example, it can relate words from the same lexeme, like "magnetism" and "magnetic".

LCS ratio works well for the case without candidate choices and significantly improve the performance of the model. This shows the importance of considering word order. However, LCS ratio helps as much for the case with candidate choices. This makes sense. Because the way we use candidate choices is simply concatenating them to queries, and it's likely that the candidate choices cannot form a meaningful sentence together with the query, and the length of LCS will remain the same with or without the candidate choices. And LCS ratio will be much lower if the candidate choices are long, which makes LCS ratio more relevant to the length of candidate choices but not with the similarity between the query and the table.

5.2.4 Error Analysis

Feature-based model tends to make mistakes when the vocabulary of two tables are similar. Figure 5.2 gives an example of such case. The predicted table and the ground truth table both contain words like "winter solstic", "December", "northern hemisphere", so the feature-based model is not able to make prediction correctly even with the help of candidate choices. However, the neural model is able to learn the semantic of queries and tables. Since the



In what part of the world does the winter solstice occur in December?

A.northern hemisphere, B.equatorial region, C.southern hemisphere, D.western hemisphere

Prediction (wrong)

	geographical region		season		date		date
In the	northern hemisphere	it is	winter	between	winter solstice; December 21	and	March Equinox; March 20
In the	northern hemisphere	it is	summer	between	summer solstice; June 21; June solstice	and	autumnal equinox; September 23
In the	northern hemisphere	it is	spring	between	March equinox; March 20	and	June solstice; summer solstice; June 21
...	...	...	...	...	...	...	...

Hemisphere/Season Relationship

Ground truth

	HEMISPHERE		ORBITAL EVENT		MONTH OF OCCURENCE
In the	northern hemisphere	, the	summer solstice	occurs in	June
In the	southern hemisphere	, the	summer solstice	occurs in	December
In the	northern hemisphere	, the	winter solstice	occurs in	December
...	...	...	...	...	...

Orbital Event Timing

Figure 5.2: An example of wrong prediction of feature-based model.

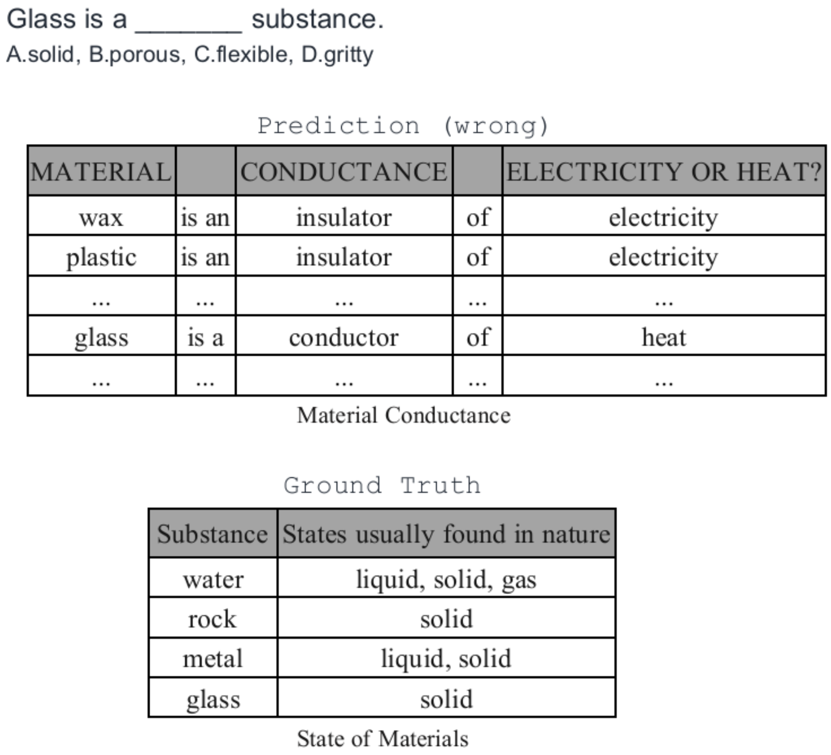
question asks about winter solstice, which is an orbital event, and the theme of the first table is season, the neural model can figure out that it's not the relevant table.

The neural model, on the other hand, suffers from inability to utilize lexical information especially when the query is short. Figure 5.3 gives an example of such case. The question asks about a property of a material, and both the ground truth and predicted tables describe properties of materials, so the neural model makes a wrong prediction. But the feature-based model can notice the word "substance", and thus it is able to predict the table correctly.

5.3 Answer Selection Experiments

5.3.1 Experiment Setup

The candidate tables are retrieved using the ensemble model with candidate choices. We use the top 1 table returned for each query as the knowledge for answer selection.



Glass is a _______ substance.
A.solid, B.porous, C.flexible, D.gritty

Prediction (wrong)

MATERIAL		CONDUCTANCE		ELECTRICITY OR HEAT?
wax	is an	insulator	of	electricity
plastic	is an	insulator	of	electricity
...	...	...	...	...
glass	is a	conductor	of	heat
...	...	...	...	...

Material Conductance

Ground Truth

Substance	States usually found in nature
water	liquid, solid, gas
rock	solid
metal	liquid, solid
glass	solid

State of Materials

Figure 5.3: An example of wrong prediction of neural model.

When preprocessing the dataset, we first remove all punctuation, and then use the tokenizer from NLTK [1] to split the queries and tables into lists of tokens. All letters are set to lower case.

The vector representations of words are initialized with GloVe word embedding by Pennington et al. [16], and are fine-tuned during training.

Baselines: To show the effectiveness of our model, we compare our model with several baselines: (1) *bag of words*: We treat the query, rows in the table, and candidate choices as bags of words. And we compare the query and a choice with each row. The choice that results in the most word overlap is chosen as the correct answer; (2) *LCS*: We treat the query, rows in the table, and candidate choices as strings. For each row in the table, we find the LCS of the row and the query, and the LCS of the row and one candidate choice, and the sum of the lengths of the two LCSs is the relevance between the row and the choice. The choice that results in the highest score with a row is considered as the correct answer; (3) *TabNN*: TabNN is a neural model proposed by Jauhar [8]. It consists of three parts: query-row mapping, choices-column mapping, cell-answer mapping. In query-row, Jauhar uses LSTM as encoder to encode the query and row into dense vectors, and the concatenation of the vector representations is fed into a fully-connected network to get the relevance score. The other two parts have the same structure; (4) *Wang et al.*: Wang et al.'s model is the state-of-the-art model on this task. Bag of words and LCS use the same knowledge as our model (top 1 table given by the table retrieval model), while TabNN doesn't do table retrieval, and Wang et al.'s uses the top 3 tables given by their own table retrieval model.

Evaluation Metric: The models are evaluated with accuracy. Accuracy is defined in Section 2.1.

Hyperparameters: We use 10% of queries in training set as validation set to tune hyperparameters. We first set the hyperparameters to default value or commonly used value, and then search for the optimal hyperparameters one by one. We apply a dropout layer to both input and output of every LSTM layer. Besides the hyperparameters for the neural network, we also need to tune the threshold θ for the answer selection algorithm to determine if a choice is good enough. The hyperparameters are given in Table 5.5.

number of filters for CNN	5
kernel size for CNN	2
state size for LSTM	64
dropout	0.2
size of hidden layer	32
θ	0.5

Table 5.5: Hyperparameters for our model.

To reduce the noise introduced by randomly generating query-table pairs for training, we set the random seed to 0. We run the models using the tuned hyperparameters for 5 times and report the **average** accuracy.

5.3.2 Performance Comparison

From the results, we find that our model outperform every baseline, and achieves 12.9% higher accuracy than previous state-of-the-art. This is be-

Model	Accuracy
TabNN [8]	56.8%
Wang et al. [31]	79.0%
Bag of words	63.8%
LCS	71.3%
Ours	$\mathbf{91.9\% \pm 0.3\%}$

Table 5.6: Accuracy of answer selection models. The results of TabNN and Wang et al. are from their original papers.

cause we optimize their method in several ways. We will discuss the influence of our modifications in the following sections.

We also find that TabNN is even worse than the bag-of-words and LCS baselines. The reason could be the poor methodology. TabNN doesn't make the effort to retrieve relevant tables before selecting the answer, and it tries to map the query to a certain row, which is error-prone.

5.3.3 Influence of Table Retrieval

Wang et al. use the top 3 tables retrieved to achieve high recall in table retrieval, but we only use the top 1 table. Here we test our answer selection model with different tables as knowledge to demonstrate the influence of table retrieval. We compare the following settings: (1) the ground truth table; (2) top 1 table generated by our table retrieval model; (3) top 2 tables generated by our table retrieval model. We don't use top 3 tables like Wang et al. because the relevant table always rank top 2 in our table retrieval model. The results are shown in Table 5.7.

Setting	Accuracy
Ground truth table	**91.9%**
Top 1 table	**91.9%**
Top 2 tables	90.9%

Table 5.7: Accuracy of answer selection given different tables.

From the results, we can see that using ground truth table and using top 1 table achieve the same accuracy. This shows the effectiveness of our table retrieval model. When using the top 2 tables, the recall of table retrieval is higher, but the accuracy decreases by 1%. This is because an extra table will result in more noise, and makes it harder for the pattern scoring model to correctly rank the patterns. Wang et al. use 3 tables as knowledge for a query, which could be one reason that their result is worse than ours.

5.3.4 Influence of Column Selection

Wang et al. score every cell in the table, while our model only scores the cells in a certain column. We conduct experiments using our answer selection model to see if column selection can improve the accuracy.

Setting	Accuracy
w/o column selection	90.9%
w/ column selection	**91.9%**

Table 5.8: Accuracy of answer selection with and without column selection. w/o stands for without, w/ stands for with.

As is shown in Table 5.8, doing column selection increases the accuracy by 1%. We already know that table retrieval helps to filter our irrelevant tables and results in higher accuracy. So it makes sense that column selection can also improve accuracy. Although selecting a wrong column may result in wrong answer selection, the system benefits more from filtering out irrelevant cells.

5.3.5 Ablation Study

To evaluate the contribution of the extra features that we add to the pattern scoring model, we do ablation study, by removing one feature from the model. The results are shown in Table 5.9.

Model	Accuracy
Full model	$91.9\% \pm 0.3\%$
-X_{feat}	$85.0\% \pm 0.9\%$
-Bilinear similarity	$91.2\% \pm 0.4\%$

Table 5.9: Ablation study of additional features.

The result shows that bilinear similarity results in about 0.7% higher accuracy. The lexical features results in 6.9% higher accuracy, which shows the

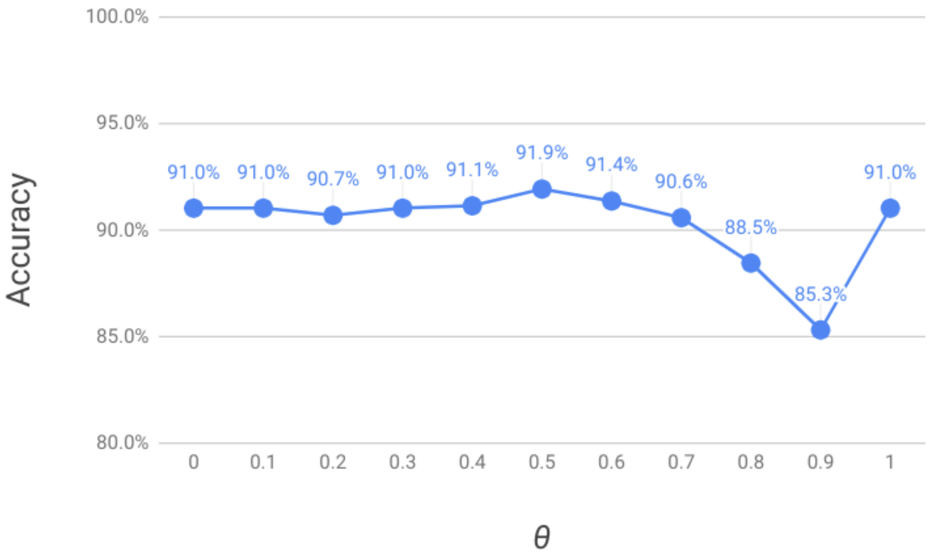


Figure 5.4: Accuracy with different θ.

importance of learning both lexical and semantic similarity. The improvement seems to unreasonably big, but actually the additional lexical features can often result in such big improvement in answer selection task. Yu et al. [39], Severyn and Moschitti [26], Tay et al. [29] all witness big absolute improvements ranging from 6% to 15%.

5.3.6 Influence of θ

We conduct experiments of answer selection using different θ to show the importance of it. Figure 5.4 shows the accuracy-θ curve.

From the curve, we can find that a proper θ can improve the accuracy by up to 6.6% comparing with improper θ. θ reflects to what extent do we believe the pattern scoring model. A lower θ makes the algorithm rely more on the pattern scoring model, because it's easy to find a cell-choice pair that matches not very well in the highest ranked answer sets. A higher θ, makes the algorithm rely less on the pattern scoring model, because a well matched cell-choice pair is not easy to be found.

Setting θ to 0 means that the correct answer is selected just based on the first answer set. And the result is good, which indicates the accuracy of our pattern scoring model. The answer selection algorithm achieves the highest accuracy when θ is 0.5. This indicates that our strategy is successful, and θ can help figure out if the pattern scoring model makes a mistake. If θ is too high, the accuracy will go down. The reason is that in this case, the pattern scoring model becomes less important, and the algorithm will often decide the correct answer based on low-rank answer sets.

It's interesting that the accuracy is the same when we set θ to 0 or 1. One possible reason is that, if a perfect-matched cell-choice pair is not found in the first answer set, then there's no perfect-matched cell-choice pair at all. When θ is 1, the algorithm cannot find a satisfying choice-cell pair in any answer set, so it will just select the best match in the first answer set.

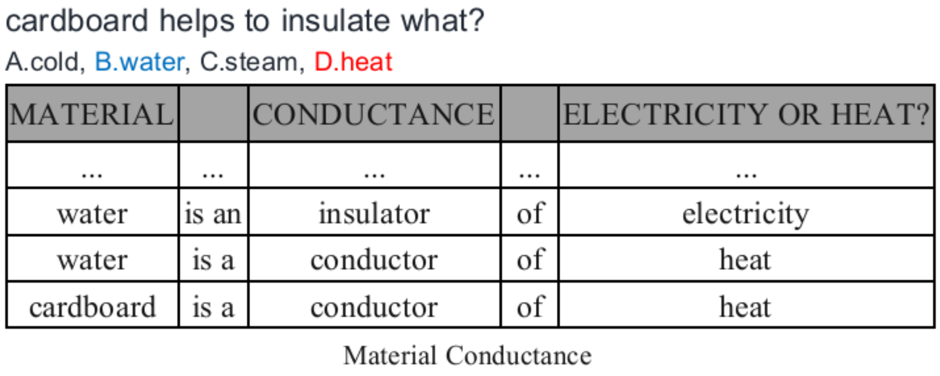
5.3.7 Error Analysis

There are several reasons that the model can select an incorrect answer.

First, there could be errors during column selection. Doing column selection results in higher accuracy, but it can result in wrong prediction, especially when the candidate choices are not extracted from the same column. Figure 5.5 shows an example of such case. "water" appears in column 0, while "heat" appears in column 4, and our column selection method wrongly selects column 0 as the relevant column, and choice B is chosen as the correct answer. This kind of error can be reduced by better column selection method. Our method only considers the candidate choices and the body of the table. If we take the query and the headers into account, the accuracy of column selection could be higher.

Second, some of the queries require additional knowledge. Figure 5.6 shows an example of such case. The correct answer is "USA", but in the table, "USA" is expressed as "the United States of America". Our model does not have the knowledge to relate "USA" to "the United States of America", and thus makes the wrong prediction. We can use techniques like entity linking to help solve this problem.

Third, some errors are caused by θ. 0.5 is the best choice for θ in our experiments, but 0.5 doesn't fit every case. In Figure 5.7, the query is very simple, but our answer selection model fails to select "Niue" because the FUZZY



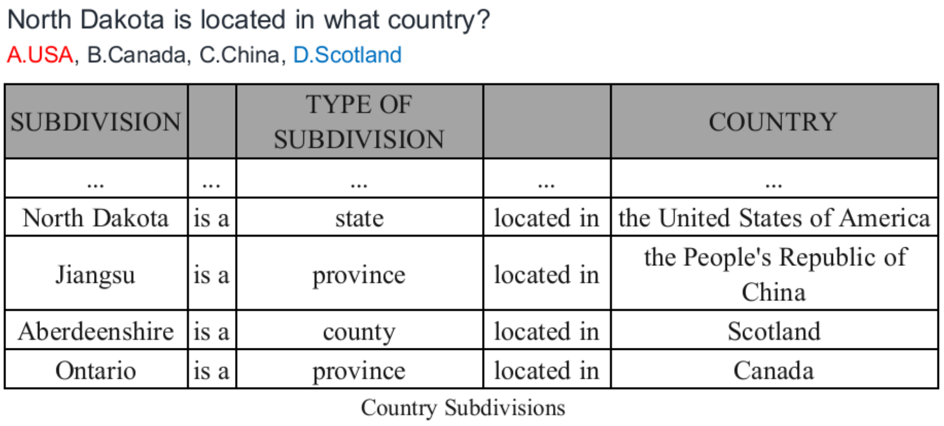
cardboard helps to insulate what?

A.cold, B.water, C.steam, D.heat

MATERIAL		CONDUCTANCE		ELECTRICITY OR HEAT?
...	...	...	...	...
water	is an	insulator	of	electricity
water	is a	conductor	of	heat
cardboard	is a	conductor	of	heat

Material Conductance

Figure 5.5: Error caused by wrong column selection. The choice marked with red is the correct answer, and the choice marked with blue is the predicted answer. It's the same for Figure 5.6-5.8.



North Dakota is located in what country?

A.USA, B.Canada, C.China, D.Scotland

SUBDIVISION		TYPE OF SUBDIVISION		COUNTRY
...	...	...	...	...
North Dakota	is a	state	located in	the United States of America
Jiangsu	is a	province	located in	the People's Republic of China
Aberdeenshire	is a	county	located in	Scotland
Ontario	is a	province	located in	Canada

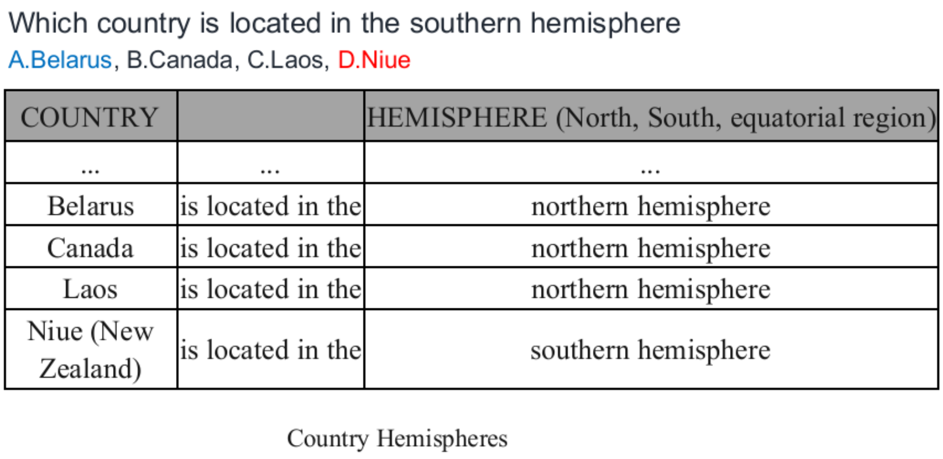
Country Subdivisions

Figure 5.6: Error caused by insufficient knowledge.

score of "Niue" and "Niue (New Zealand)" is only 0.36, which is lower than the threshold. So the second answer set is used to select the answer, and the prediction is wrong.

Fourth, the pattern scoring model may fail to rank patterns correctly. This could be alleviated by training the model on a larger dataset

Fifth, there are some faults in the dataset. Sometimes more than 1 choices are correct, but only one of them is marked as correct answer. And sometimes the query itself is incorrect. In Figure 5.8, the question should be "*On average, what weighs 28500 kilograms?*". Missing an "8" leads to wrong prediction.



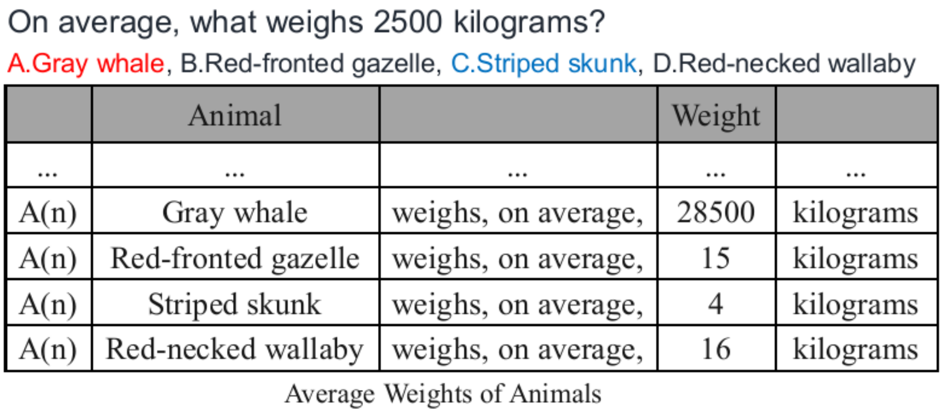
Which country is located in the southern hemisphere

A.Belarus, B.Canada, C.Laos, D.Niue

COUNTRY		HEMISPHERE (North, South, equatorial region)
...	...	...
Belarus	is located in the	northern hemisphere
Canada	is located in the	northern hemisphere
Laos	is located in the	northern hemisphere
Niue (New Zealand)	is located in the	southern hemisphere

Country Hemispheres

Figure 5.7: Error caused by θ.



On average, what weighs 2500 kilograms?

A.Gray whale, B.Red-fronted gazelle, C.Striped skunk, D.Red-necked wallaby

	Animal		Weight	
...	...	...	...	...
A(n)	Gray whale	weighs, on average,	28500	kilograms
A(n)	Red-fronted gazelle	weighs, on average,	15	kilograms
A(n)	Striped skunk	weighs, on average,	4	kilograms
A(n)	Red-necked wallaby	weighs, on average,	16	kilograms

Average Weights of Animals

Figure 5.8: Error caused by faults in the dataset.

5.4 Summary

In this chapter, we evaluate our models on a publicly available dataset, TabMCQ. And our QA system achieves 12.9% higher accuracy than previous state-of-the-art.

The ablation study of feature-based table retrieval model shows the effectiveness of our novel features. And the ensemble of feature-based and neural model performs the best. Experiments on answer selection shows that our table retrieval model works pretty well, and using the tables retrieved by the model results in the same accuracy as using the ground truth tables.

In the experiments on answer selection, we find that the biggest improvement comes from the lexical features we add to the pattern scoring model, which confirms that the combination of lexical and semantic matching will be better than semantic matching alone. Experiments also show that all of our modifications on Wang et al.'s model improve the accuracy.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we explore the task of multiple-choice question answering over semi-structured tables. We tackle the task by two steps: table retrieval and answer selection. We evaluate our model on the TabMCQ dataset, and the results show that our model achieves the state-of-the-art performance.

To retrieve relevant tables, we build a feature-based model that focuses on lexical similarity. If we retrieve the relevant table only with the query, Wang et al.'s neural model performs as good as ours. However, our model can effectively utilize the additional information provided by the candidate choices. As a result, if we take candidate choices into account, our feature-based model outperforms Wang et al.'s neural model by more than 2% (improvements reported in this chapter are absolute improvement). And the combination of the feature-based model and neural model results in even better performance. The improvement is around 2% when we do not use candidate choices, and 0.1% when we use candidate choices. Besides, our two novel features, FUZZY and LCS ratio, are shown to be effective. They can bring an absolute improvement up to 0.8%.

To select the correct answer from the candidate choices, we improve Wang et al.'s method in several aspects. First, we do column selection to filter out irrelevant cells in the table. Second, to more accurately score patterns, we add lexical features and bilinear similarity of the representations of query and pattern to the pattern scoring model. Third, we develop an effective answer selection algorithm. Experimental results show that our modifications are effective. Doing column selection improves the accuracy by 1%. The additional lexical features and bilinear similarity result in an improvement of 6.9% and 0.7% respectively.

Our system is also robust to minor changes in the tables. If some new rows are added to the tables, we do not need to re-train everything. For table retrieval models, the features in the feature-based model are extracted in an unsupervised way, and the fully-connected network that computes the score is not influenced by the body of the tables; the neural model only use the "link words" in the table body, so it's not influenced by the table body too. The pattern scoring model in answer selection is only guaranteed to be trained on rows that are relevant to training queries, so adding more rows to the tables won't require re-training. But if the changes are massive, and many tables are added to the dataset, we need to re-train or at least fine-tune the system on the new dataset to ensure good performance.

To conclude, we answer the two questions we proposed in the thesis statement. For table retrieval, a feature-based model that focuses on lexical similarity can perform comparably well with a neural model. And if we take the candidate choices into account, the feature-based model can perform even better than a neural model. For answer selection, taking lexical similarity into account can result in huge improvement.

6.2 Future Work

There are still much to be done in future research.

We only use fixed word embedding to represent words, and basic LSTM and BiLSTM to encode sentences in the neural table retrieval model and pattern scoring model. We can try more advanced word embeddings (e.g., ELMo [17]) and sentence encoders (e.g., DiSAN [27] and BERT [4]) to improve the performances of the models.

In the neural table retrieval model, we only encode the table caption, table headers and a small part of the table body. We can also try to encode the whole table body. One possible way is to encode each row of the table separately in to a vector, and then use another neural network (like LSTM) to encode the vector representations of each row to get the representation for the table body.

In the answer selection model, we first compare the query to the rows of the relevant table to get relevant cells, and then match the candidate choices with the relevant cells. We can try to rephrase the query and one candidate choice into a statement first, and then directly match the statement with rows in the table.

Another important issue is only 63 domain-specific tables are involved in our experiments. If we want to build a more general table retrieval method, we need to study a larger table collection. And we can even build our own dataset. We can search the web for raw text corpora, and extract similar sentences by pattern matching to build the semi-structured tables. For example, we can extract sentences in the pattern "xx(someone) is the president of xxx(some country)" to build a table about presidents. However, writing queries related to the tables requires manual effort.

Our system consists of two parts, table retrieval and answer selection, and the two parts are trained separately. We can also try to build an end-to-end model. But this will require more training data, and we need to carefully design the loss function so that the whole system can be properly optimized. And it will also be harder to do preprocessing like column selection in an end-to-end model.

We can also explore question answering without candidate choices. Our system is actually capable of doing this. But our simple column selection method will not work without candidate choices. One way to do column selection in this scenario is to compare the query with the headers of the body. Or we can just skip the column selection and score every cell like Wang et al. And because we do not have candidate choices to refer to, we have to trust our pattern scoring model, and just use the every cell in the first answer set as the answer for the question.

Besides, our QA system is not equipped with inference ability, and cannot relate entities expressed in different ways. We can try to augment our system so that it can answer queries that require inference from several tables or documents.

In this thesis, we focus on IR-based QA. We can also try to build a knowledge-based QA system to solve the task. We can also explore other QA tasks, like SQuAD [20], WikiSQL [42], etc.

References

[1] S. Bird, “NLTK: the natural language toolkit,” in *ACL*, The Association for Computer Linguistics, 2006. 35, 40

[2] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, “Webtables: Exploring the power of tables on the web,” *PVLDB*, vol. 1, no. 1, pp. 538–549, 2008. 9, 14, 15

[3] C. J. Date and H. Darwen, *A Guide to SQL Standard, 4th Edition.* Addison-Wesley, 1997. 2

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018. 50

[5] B. F. Green Jr, A. K. Wolf, C. Chomsky, and K. Laughery, “Baseball: An automatic question-answerer,” in *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, ACM, 1961, pp. 219–224. 1

[6] W. He, K. Liu, J. Liu, Y. Lyu, S. Zhao, X. Xiao, Y. Liu, Y. Wang, H. Wu, Q. She, X. Liu, T. Wu, and H. Wang, “Dureader: A chinese machine reading comprehension dataset from real-world applications,” in *QA@ACL*, Association for Computational Linguistics, 2018, pp. 37–46. 1

[7] M. Heilman and N. A. Smith, “Tree edit models for recognizing textual entailments, paraphrases, and answers to questions,” in *HLT-NAACL*, The Association for Computational Linguistics, 2010, pp. 1011–1019. 11

[8] S. K. Jauhar, “A relation-centric view of semantic representation learning,” PhD thesis, Carnegie Mellon University, 2017. 13, 25, 41, 42

[9] S. K. Jauhar, P. D. Turney, and E. H. Hovy, “Tables as semi-structured knowledge for question answering,” in *ACL (1)*, The Association for Computer Linguistics, 2016. 4

[10] ——, “Tabmcq: A dataset of general knowledge tables and multiple-choice questions,” *CoRR*, vol. abs/1602.03960, 2016. 6

[11] V. Jijkoun, M. de Rijke, *et al.*, “Recognizing textual entailment using lexical similarity,” in *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, Citeseer, 2005, pp. 73–76. 11

[12] D. Jurafsky and J. H. Martin, "Speech and language processing," 1, 3

[13] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. H. Hovy, "RACE: large-scale reading comprehension dataset from examinations," in *EMNLP*, Association for Computational Linguistics, 2017, pp. 785–794. 13

[14] T. M. Lai, T. Bui, and S. Li, "A review on deep learning techniques applied to answer selection," in *COLING*, Association for Computational Linguistics, 2018, pp. 2132–2144. 11

[15] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-structured tables," in *ACL (1)*, The Association for Computer Linguistics, 2015, pp. 1470–1480. 4

[16] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, ACL, 2014, pp. 1532–1543. 35, 40

[17] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *NAACL-HLT*, Association for Computational Linguistics, 2018, pp. 2227–2237. 50

[18] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *PVLDB*, vol. 5, no. 10, pp. 908–919, 2012. 9, 14, 15

[19] T. Qin, T. Liu, J. Xu, and H. Li, "LETOR: a benchmark collection for research on learning to rank for information retrieval," *Inf. Retr.*, vol. 13, no. 4, pp. 346–374, 2010. 15

[20] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *EMNLP*, The Association for Computational Linguistics, 2016, pp. 2383–2392. 2, 52

[21] M. Richardson, C. J. C. Burges, and E. Renshaw, "Mctest: A challenge dataset for the open-domain machine comprehension of text," in *EMNLP*, ACL, 2013, pp. 193–203. 13

[22] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, "Okapi at TREC-3," in *TREC*, vol. Special Publication 500-225, National Institute of Standards and Technology (NIST), 1994, pp. 109–126. 10, 18

[23] S. Ruder, *Nlp progress-question answering*, `https://nlpprogress.com/english/question_answering.html`. 2

[24] M. Sachan, K. A. Dubey, E. P. Xing, and M. Richardson, "Learning answer-entailing structures for machine comprehension," in *ACL (1)*, The Association for Computer Linguistics, 2015, pp. 239–249. 13

[25] C. N. dos Santos, M. Tan, B. Xiang, and B. Zhou, "Attentive pooling networks," *CoRR*, vol. abs/1602.03609, 2016. 12

[26] A. Severyn and A. Moschitti, "Learning to rank short text pairs with convolutional deep neural networks," in *SIGIR*, ACM, 2015, pp. 373–382. 5, 30, 44

[27] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, “Disan: Directional self-attention network for rnn/cnn-free language understanding,” in *AAAI*, AAAI Press, 2018, pp. 5446–5455. 50

[28] M. Tan, B. Xiang, and B. Zhou, “Lstm-based deep learning models for non-factoid answer selection,” *CoRR*, vol. abs/1511.04108, 2015. 12

[29] Y. Tay, M. C. Phan, A. T. Luu, and S. C. Hui, “Learning to rank question answer pairs with holographic dual LSTM architecture,” in *SIGIR*, ACM, 2017, pp. 695–704. 30, 44

[30] Y. Tay, L. A. Tuan, and S. C. Hui, “Hyperbolic representation learning for fast and efficient neural question answering,” in *WSDM*, ACM, 2018, pp. 583–591. 11

[31] H. Wang, X. Zhang, S. Ma, X. Sun, H. Wang, and M. Wang, “A neural question answering model based on semi-structured tables,” in *COLING*, Association for Computational Linguistics, 2018, pp. 1941–1951. 4, 10, 13, 14, 21, 24, 26,

[32] M. Wang and C. D. Manning, “Probabilistic tree-edit models with structured latent variables for textual entailment and question answering,” in *COLING*, Tsinghua University Press, 2010, pp. 1164–1172. 11

[33] M. Wang, N. A. Smith, and T. Mitamura, “What is the jeopardy model? a quasi-synchronous grammar for QA,” in *EMNLP-CoNLL*, ACL, 2007, pp. 22–32. 11

[34] S. Wang, M. Yu, J. Jiang, and S. Chang, “A co-matching model for multi-choice reading comprehension,” in *ACL (2)*, Association for Computational Linguistics, 2018, pp. 746–751. 13

[35] Z. Wang, W. Hamza, and R. Florian, “Bilateral multi-perspective matching for natural language sentences,” in *IJCAI*, ijcai.org, 2017, pp. 4144–4150. 12

[36] Z. Yan, D. Tang, N. Duan, J. Bao, Y. Lv, M. Zhou, and Z. Li, “Content-based table retrieval for web queries,” *CoRR*, vol. abs/1706.02427, 2017. 10, 14, 15, 36

[37] X. Yao, B. V. Durme, C. Callison-Burch, and P. Clark, “Answer extraction as sequence tagging with tree edit distance,” in *HLT-NAACL*, The Association for Computational Linguistics, 2013, pp. 858–867. 11

[38] W. Yin, S. Ebert, and H. Schütze, “Attention-based convolutional neural network for machine comprehension,” *CoRR*, vol. abs/1602.04341, 2016. 13

[39] L. Yu, K. M. Hermann, P. Blunsom, and S. Pulman, “Deep learning for answer sentence selection,” *CoRR*, vol. abs/1412.1632, 2014. 11, 31, 44

[40] M. D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, vol. abs/1212.5701, 2012. 21, 24, 31

[41] S. Zhang and K. Balog, “Ad hoc table retrieval using semantic similarity,” in *WWW*, ACM, 2018, pp. 1553–1562. 9, 14, 15, 17

[42] V. Zhong, C. Xiong, and R. Socher, “Seq2sql: Generating structured queries from natural language using reinforcement learning,” *CoRR*, vol. abs/1709.00103, 2017. 2, 4, 52

[43] H. Zhu, F. Wei, B. Qin, and T. Liu, “Hierarchical attention flow for multiple-choice reading comprehension,” in *AAAI*, AAAI Press, 2018, pp. 6077–6085. 13

Appendix A

Neural Networks

A.1 Feed-forward Neural Network

A neural network has many simple units called neurons. A feed-forward neural network is a neural network where the connections between the neurons do not form a cycle. Figure A.1 shows a neural network.

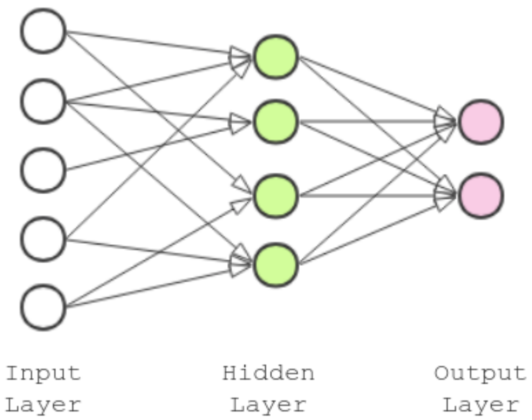


Figure A.1: A feed-forward neural network.

A feed-forward neural network has three types of layers: input layer, hidden layer and output layer. Each layer consists of several neurons. And the incoming arrow denotes the input to the neuron, and outgoing arrow denotes the output of the neuron. If every neuron in a layer is connected to every neurons in the next layer, then this layer is called a fully-connected layer.

If we denote the input as $\mathbf{x}$, and the output as $\mathbf{y}$, then the feed-forward neural network can be described as a mapping function $\mathbf{y} = f(\mathbf{x})$. A fully-connected layer performs a vector-matrix multiplication: $\mathbf{h} = W\mathbf{x}$, where W is a transformation matrix. Then people usually apply a non-linear function to $\mathbf{h}$. The non-linear function is also known as activation function. Commonly used activation functions include tanh, relu and sigmoid. The computation is executed layer by layer until the output layer.

A.2 Convolutional Neural Network

Convolutional Neural Network (CNN) is actually a special case of feed-forward neural network. The idea of CNN is to reduce the connections between the input layer and hidden layer. In a fully-connected network, a neuron is connected to every neuron in adjacent layers. But in CNN, each neuron in hidden layer only processes a subset of the input.

In natural language processing, the input to a CNN is usually a sequence X with m words. And the i-th word x_i can be represented with a d-dimensional vector $\mathbf{x}_i$. Let $\mathbf{c}_i = [\mathbf{x}_{i-n+1} \oplus \mathbf{x}_{i-n+2} ... \oplus \mathbf{x}_i]$, where $\oplus$ denotes concatenation, and n is the filter size of the CNN. The convolution layer does the following transformation to generate the representation for the n-gram $x_{i-n+1}, x_{i-n+2}, ..., x_i$:

$$\mathbf{p}_i = tanh(W\mathbf{c}_i + \mathbf{b}) \tag{A.1}$$

where $\mathbf{p}_i \in \mathbb{R}^{d'}$, and $W \in \mathbb{R}^{d' \times nd}$ is the convolution matrix, and bias $\mathbf{b} \in \mathbb{R}^{d'}$. Then the representation of input sequence $\mathbf{X} \in \mathbb{R}^{d'}$ is generated by maxpooling over all n-gram representations:

$$\mathbf{X}_j = \max(\mathbf{p}_{1,j}, \mathbf{p}_{2,j}, ...)(j = 1, ..., d') \tag{A.2}$$

A.3 Recurrent Neural Network

In feed-forward network, the output of each layer will always be passed to next layer. But in recurrent neural network (RNN), the output of a layer can be fed back to the layer itself as input. RNN is suitable to deal with sequences, and can map a sequence of any size into a fixed-size vector.

Long Short-Term Memory (LSTM) is a prevailing RNN type. It models the word sequence X as follows:

$$\mathbf{f}_t = sigmoid(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{A.3}$$

$$\mathbf{i}_t = sigmoid(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{A.4}$$

$$\mathbf{o}_t = sigmoid(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{A.5}$$

$$\mathbf{p}_t = \mathbf{f}_t \circ \mathbf{p}_{t-1} + \mathbf{i}_t \circ tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{A.6}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ tanh(\mathbf{p}_t) \tag{A.7}$$

where $\circ$ denotes element-wise product, $\mathbf{x}_t \in \mathbb{R}^d$ is the vector representation for word x_t, $\mathbf{f}_t \in \mathbb{R}^h$ is the forget gate, $\mathbf{i}_t \in \mathbb{R}^h$ is the input gate, $\mathbf{o}_t \in \mathbb{R}^h$ is the output gate, $\mathbf{h}_t \in \mathbb{R}^h$ is the hidden state vector, $\mathbf{p}_t \in \mathbb{R}^h$ is the cell state vector that can memorize the history of the input sequence, $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ are weight matrices, and $\mathbf{b} \in \mathbb{R}^h$ is bias vector. People often use the final hidden state $\mathbf{h}_T$ to represent the input sequence.