Question Answering from Web Extracted Tables

Students: Bhavya Karki, Fan Hu, Nithin Haridas, Suhail Barot, Zihua Liu
Mentors: Anthony Tomasic, Matthias Grabmair, Lucile Callebert

ABSTRACT

Getting answers from structured data using natural language is an active research topic in the domain of question answering. It is difficult to obtain knowledge bases that are tailored for the problem of question answering from tabular data. We propose a dataset that contains some of the most commonly asked questions to home devices and their corresponding answers found in tables in a website. Our dataset is novel in a way that it includes two kinds of tables: entity-instance table and the key-value table. Each set of data in train and test dataset comprises of a table of either kind, a natural language query and a corresponding structured query (SQL). We divide our problem into several sub-problems, and conduct experiments to arrive at the correct answer in the table. The sub-problems include information retrieval and machine learning classification. We extract various kinds of features specific to each sub-problem. With these sub-problems in place, we aim to stitch together to form a complete solution, i.e. to construct the SQL query from its parts. The work provides qualitative results and error analysis for each sub-problem to help identify future improvement areas.

1. INTRODUCTION

Question answering systems have proved to be useful technology. However performance is poor when generating structured queries, a major reason being that it is hard to generate large structured datasets. Generating structured query language (SQL) from a natural language question and semantic parsing of structured data are issues that have recently attracted researchers' attention. The state of the art solutions to question answering of structured queries generally uses neural models. Neural models require datasets large enough for the network parameters to learn the model. Our pilot testing of neural models indicated that we have insufficient data for applying sequence-to-sequence (seq2seq) models that encode a natural language question and produce a structured query as the output. We instead decompose the problem into individually crafted sub-problems to throw some light on the difficulties of this class of applications. With additional data, we intend to return to seq2seq approaches. In this paper, we analyze the reasoning involved in constructing structured queries and build models to predict each step. These steps are directly related to the parsing of a structured query into its composing components such as the SELECT clause, WHERE clause and so on. We then combine these steps to generates queries that has much lower requirements for size of the dataset.

This approach offers a deeper understanding of the reasoning processes used to generate structured queries. In particular we directly identify the *semantic* information, in addition to the syntactic information, required for evidence generation of a structured query. This information is critical to guiding future efforts in training set construction, merging with existing knowledge sources, and integration with structured sources. Tabular information includes regular tables, structured grids of information, lists of information, and property-value tables. This paper provides several contributions:

- a new dataset generated via crowd sourcing that contains less internal bias.
- decomposing the big problem into several individually crafted sub-problems and providing insights on each sub-problem.
- experimental results that deal with issues pertaining to limited data by extensive feature engineering and application of machine learning models in an intelligent manner.

2. HYPOTHESIS

This system is constructed with the assumption that the tables contain information sufficient to answer the question. The framework is constructed in such a way that a sequence of steps will lead to identifying the location of the answer in the table. The underlying confidence is that it is possible to develop a question answering system for structured data and that it is possible to retrieve the relevant information using SQL query for a given question. The proposed framework targets to establish a baseline performance for this kind of dataset.

3. RELATED WORK

There have been a good number of literature that address the task of structured question answering. One of the earlier works in using structured data as knowledge sources (Jauhar et al.) [5] focuses on exploring tables as semi-structured knowledge for multiple-choice question (MCQ) answering. The tables contain text in free form and are mostly general knowledge based facts. They contain answers to more than 9000 crowd-sourced questions in their base dataset AI2's Aristo Tablestore. The authors develop a feature-driven model that uses these MCQs to perform question answering, while performing fact checking and reasoning over tables. Our models heavily depend on features of the text that arise out of data in tables. We identify and extract features to use for every sub-problem, therefore we can consider our models to be feature driven as well. (Hayati et al.) [4] use semantic and n-gram similarity for retrieval. They introduce

ReCode, a sub-tree retrieval based framework for neural code generation using natural language. They use dynamic programming based methods to retrieve semantically similar sentences, then construct the syntax tree using n-gram of action sequences. The importance of semantic similarity features was noted and we experiment with incorporating semantic similarity features in our models. (Cheng et al) [2] introduce a neural semantic parser that uses predicate-argument structures to convert natural language utterances to intermediate representations. These representations are then mapped to target domains. Similarly, we break down most of our questions into a projection column and conditional row to construct the structured query.

One of the most closely related prior work was the curation of WikiSQL (Zhong et al) [9] and the proposal of Seq2SQL, a deep neural network for translating natural language questions to corresponding SQL queries. WikiSQL consists of 80654 hand-annotated examples of questions and SQL queries distributed across 24241 tables from Wikipedia, which is an order of magnitude larger than other comparable datasets. The dataset split ensures that the test and train sets are disjoint so that there are only unseen examples during test times. The Seq2SQL model uses rewards from in-the-loop query execution over the database to learn a policy to generate the query. It is a pointer network that limits the output space of the generated sequence to the union of the table schema, question utterance, and SQL keywords. Our dataset is similar to the dataset WikiSQL, but we distinguish between the types of tables covered in the dataset. Inspired by this model, early efforts in this project were directed towards sequence to sequence models but failed due to limitations of data. Hence, we break down the approach into sub-problems that can be solved with machine learning models.

4. DATA COLLECTION

In this section we describe the data collection process. Data collection was conducted in two phases that are descried in the following subsections: 1) collect questions from real users who looked for answers on the web, then 2) find the answer in tabular data, extract the corresponding table and write an SQL query to retrieve the answer from the table.

4.1 Collecting questions

To generate our dataset, we created Amazon Mechanical Turk (AMT) tasks that asked workers about "5 questions you recently had where you searched for answers on the web"[1]. Workers were asked to generate questions from their past by going through their browser history, as opposed to other generation methods [8, 5] where workers are primed to ask questions based on a given table. Note that we also did *not* limit workers to questions that have an answer in tabular form. Collecting question that emerged in real life situations allows us to generate a dataset that is close to what people are actually interested in. Each worker was asked for 5 triples composed of (i) a natural language question, (ii) the URL of a page that contained the answer to the question, and (iii) the answer to the question. Workers were paid $1.75 upon completion of the task. In a pre-study we observed that we were likely to collect form workers a large amount of data about sports and weather, leading to a large bias in data collection. We therefore amended the task description to ban weather and sports questions.

4.2 Collecting tables and SQL queries

After collecting questions, we need to collect tables that contain the answer to our questions, as well as SQL queries that retrieve the answer from those tables. This was done in a second and separate step of data collection. As this step requires some expertise (i.e. identifying tabular data) as well as skill and knowledge (i.e. extracting a table and writing an SQL query), we did not use AMT but recruited several students from the Computer Science Department to perform this task. After going through detailed instructions with one team member of the project, the student worked up to 10 hours a week for three months and were paid according to CMU's guidelines.

The process for collecting tables and SQL queries is the following: for each new question (previously collected through AMT), 1) search the web for a page with tabular data containing the answer. If not such page can be found, the question is discarded. 2) Extract the relevant table from that page. Workers used two different tools to extract tables: import.io[2] and SmartWrap [3]). 3) Write and execute a SQL query to extract the answer from the table.

This process of collecting tables and SQL queries is extremely time consuming, which is why our dataset is very limited in size.

4.3 SQL query and $\sim$ operator

Inspired by the work of [1], we introduced a new SQL operator, the $\sim$ operator. The objective is to facilitate the matching between terms occurring in the question and terms occurring in the table. Indeed, because the questions from our dataset were not generated while looking at a table, we observe differences between the vocabulary used in the question and in the table. An example of a query using the $\sim$ is : `SELECT` ... `FROM` ... `WHERE column` $\sim$ `question_word`. The $\sim$ operator first tries to find an exact match for `question_word` in `column` (using the SQL = operator), then tries to find an approximate match (using the SQL `LIKE` operator) and finally uses word vectors[3] to find a term semantically close to `question_word`, as proposed by [1].

4.4 Dataset Analysis

We collected 302 questions each paired with their own table, SQL query and expected answer. We further divided the dataset into a train set (238 examples) and a test set (64 examples). Each question is unique but some questions might be semantically equivalent (e.g. *When is easter this year?* and *What day is easter on this year?*) or very close (e.g. *What is the capital of Louisiana?* and *What is the capital of Portugal?*), so one question might be answered using a different table than the one it specifically is paired with. All questions can be answered via a unique table (i.e. the SQL `JOIN` operator never appears in our dataset).

Further analyzing the dataset, we distinguished two categories of tables: entity-instance tables and key-value tables. An entity instance table contains information about several entities (e.g. Figure 1 is an entity instance table, it contains the presidency information of different presidents in the history of the USA), while a key-value table is related to a single entity (e.g. Figure 2 is a key-value table and it shows some basic information of Donald Trump).

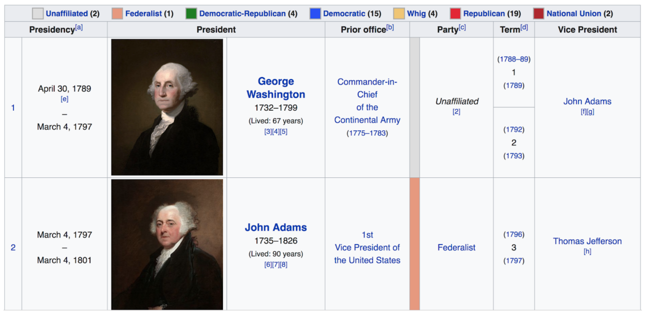
5. DEVELOPMENT GOALS

At an abstract level, our goal for this semester was to explore the approach explained in details in the following sections and using the results from our experiments to judge its feasibility.

[1] `https://cmu-rerc-apt.github.io/QASdatacollection/examples.html`

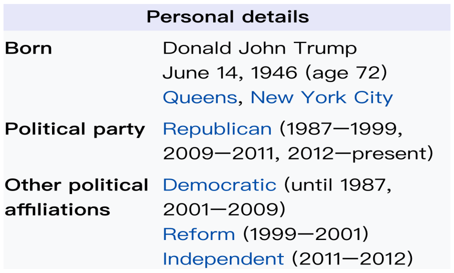
[2] commercially available at: `https://www.import.io/`

[3] We used word vectors pre-trained on Google news available at `https://code.google.com/archive/p/word2vec/`.



⬜ Unaffiliated (2) 🟧 Federalist (1) 🟩 Democratic-Republican (4) 🟦 Democratic (15) 🟨 Whig (4) 🟥 Republican (19) 🟥 National Union (2)								
Presidency[a]		President		Prior office[b]	Party[c]		Term[d]	Vice President
1	April 30, 1789 [e] – March 4, 1797		George Washington 1732–1799 (Lived: 67 years) [3][4][5]	Commander-in-Chief of the Continental Army (1775–1783)	⬜	Unaffiliated [2]	(1788–89) 1 (1789)	John Adams [f][g]
							(1792) 2 (1793)	
2	March 4, 1797 – March 4, 1801		John Adams 1735–1826 (Lived: 90 years) [6][7][8]	1st Vice President of the United States	🟧	Federalist	(1796) 3 (1797)	Thomas Jefferson [h]

Figure 1: An example of entity-instance table



Personal details	
Born	Donald John Trump June 14, 1946 (age 72) Queens, New York City
Political party	Republican (1987—1999, 2009—2011, 2012—present)
Other political affiliations	Democratic (until 1987, 2001—2009) Reform (1999—2001) Independent (2011—2012)

Figure 2: An example of key-value table

At the start of the semester, we decided to break down the problem to increase its granularity. As listed below, we came up with 7 sub-problems that we planned to solve, tackling one at a time in a sequential manner.

- **Sub-problem 1: Table Type Recognition**
- **Sub-problem 2: Table Transpose**
- **Sub-problem 3: Source Selection**
- **Sub-problem 4: Complete SELECT Clause**
- **Sub-problem 5: Complete WHERE Clause**
- **Sub-problem 6: Complete ORDER BY Clause**
- **Sub-problem 7: Complete LIMIT Clause**

Given the granularity of our new problem definition, we made a schedule that every sub-problem was assigned a particular amount of time, usually one or two weeks, depending on our analysis of the length and complexity of the sub-problem.

We completed detailed experimentation and error analysis for the first 5 sub-problems this semester. We found that some sub-problems were more involved than we anticipated, and this resulted in us investing more time in them than we had originally planned.

We also planned to work on sub-problems 6 and 7 initially. However, as we found that there are very limited relevant data in the current dataset to build a solid model, we decided to continue working on these 2 sub-tasks after we collect more data in the future.

6. SYSTEM DESIGN OVERVIEW

Our approach to the question answering problem follows a granular approach. We divide the problem into sub-problems based on the structure of the SQL query desired. As shown in Figure 3, we build the SQL query step by step, passing along key information from one step to the next in the pipeline. The details of each step are described in Section 7.1.

In the first step, we identify the type of each table, and classify it as either entity-instance or key-value. Once we accomplish this, we then use this information in the second step, where we transpose all the key-value type tables into entity-instance type. Now that we have a more homogeneous structure, all the tables are passed as an input to the next step in the pipeline.

Working on those pre-processed tables, for each question, we try to determine which table contains the answer. After finding it, we pair the question and table together as a question-table pair. We then have the FROM clause for the query. In the fourth step, we focus on extracting which column(s) from the table need to be present in the query, i.e. the SELECT clause of the query. Given the correct columns, we move to the fifth step. Here, we want to predict the correct row(s), i.e. the WHERE clause of the query. We now have the correct row(s) and column(s) for the query.

After the first 5 steps, we should complete the full SQL query for most questions. However, some questions are such that they require to be ordered by a certain column, or require a fixed number of rows as the answer. For example, *Who is the current President of USA?* needs to be ordered by year, and *Who are the top 5 goal scorers in La Liga?* needs a specific number of rows. Hence, the next steps are to determine the columns used for sorting (i.e. the ORDER BY clause of the query) and the number of rows to be displayed.(i.e. the LIMIT clause of the query).

After performing all these steps, we obtain all the clauses needed to construct a SQL query which corresponds to the given question, we could then obtain answers from the appropriate table using this formed SQL query.

7. EXPERIMENTAL DESIGN

As discussed in Section 5, we divided our entire pipeline into several sub-problems. For each sub-problem, we analyzed our dataset, extracted features and built an individual model to solve the problem. We used different metrics to evaluate the performance of model for each sub-problem.

7.1 Machine Learning Models/Algorithms

7.1.1 Table Type Recognition

The type of the table strongly informs the content of the SQL query, so we train different classification models such as decision tree and logistic regression with the following features to classify tables as entity-instance or key-value tables:

- **The number of columns in the table**
  In most cases, key-value tables would only contain 2 columns, a "key" column and a "value" column. If the number of columns in a table is equal to 2, then it is with great possibility that this table is a key-value table.

- **The number of columns in the table when ignoring link columns**
  We ignored link columns in the table and re-calculate the column number to increase the robustness of the aforementioned first feature.

- **The presence of "key" or "property" in the columns headers**
  After analyzing our dataset, we found that most of the key-value tables contain keyword "key" or "property" in the header of "key" column.

- **The normalized variation of content length in terms of number of words**
  To get this feature, we first calculated the number of words within each cell by using white-space as the separator. We

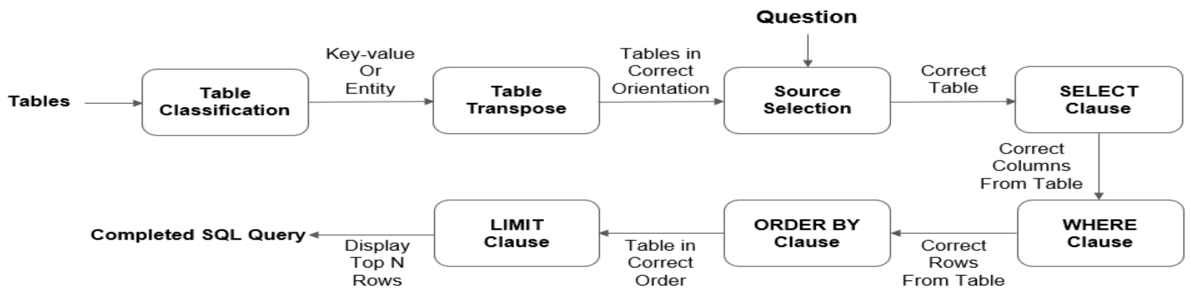


Figure 3: System Design Overview

Key	Value
spouse	Melania Trump (m. 2005), Marla Maples (m. 1993-1999), Ivanka Trump (m. 1977-1992)
born	June 14, 1946 (age 71 years), Jamaica hospital medical center, New York city, NY
height	6' 3"
net worth	3.1 billion USD (2018)
education	Wharton School of the University of Pennsylvania (1966-1968) , more

Table 1: Original key-value table

spouse	born	height	net worth	education
Melania Trump (m. 2005), Marla Maples (m. 1993-1999), Ivanka Trump (m. 1977-1992)	June 14, 1946 (age 71 years), Jamaica hospital medical center, New York city, NY	6' 3"	3.1 billion USD (2018)	Wharton School of the University of Pennsylvania (1966-1968) , more

Table 2: Entity-instance table after transposing

then normalized the original cell length by dividing lengths of the cells within the same column by the maximum cell length of that column. The variation is calculated with the normalized length. For key-value table, the cells within the same column are different attributes of a single entity. Most of the time there is no uniform pattern for the content of the cells. Therefore, the cell lengths of the same column tend to be versatile for key-value tables. While for entity-instance tables, the content length of each cell in the same column tends to be similar, because each column of the entity-instance table corresponds to a specific attribute and it usually has a similar format in content. If each column in a table has similar length, then it raises the possibility of the table being classified as entity-instance table.

- **The normalized variance of presence of digits**
  The function of this feature is similar to the previous one. Since each column of the entity-instance tables corresponds to a specific attribute, the pattern of the presence of digits would be similar for each cell within the table. As we can be seen from Figure 1, the first column is the presidency period of the president, and each cell of that column contains two year description and two day description in digits, so that every cell in that column would contain digits. However, for key-value tables, since cells in the same column correspond to different attributes, the uniform pattern of the presence of digits does not hold. If none of the cells or all the cells contain digits for each column, the variance would equal to zero. The higher the value of this feature is, the more random the presence of digits in a column is. Since the columns of entity-instance tables are more likely to have a uniform pattern of the presence of the digits, a higher variance in the presence of digits is an indicator that the table has higher probability be a key-value table.

7.1.2 Tables Transpose

As mentioned in above section, we divide the tables in our dataset into two categories, entity-instance tables and key-value tables. Since we classify these two types of tables based on the number of entities it contains, we found that key-value tables could be converted into entity-instance tables through transpose. Table 1 shows an original key-value table that contains some basic information of Donald Trump. Through transpose, it is converted into an entity-instance Table 2. For key value tables, the goal of the SQL query is to find the target row, while for the transposed entity-instance table, the goal is to find the target column. For example, the ground truth SQL query for Table 1 is `"SELECT Value FROM Donald-Trump WHERE Key~ birthday"`, and the ground truth SQL query for table 2 is `"SELECT born FROM Donald-Trump "` (`Donald-Trump` is the table name). Therefore, through table transpose, we converted all key-value tables to entity-instance tables to maintain conciseness and consistency of the following experiments. Our following algorithms only need to consider one type of tables.

7.1.3 Source Selection

To answer a specific question, we first need to correctly identify the source table in the entire dataset corresponding to that question. Since the source table tends to contain the words occurring in its corresponding question, we calculated the term frequency-inverse document frequency (TF-IDF) of each word in each question and each table, and mapped the questions as well as tables to a vector space. For each question, we chose the table that has the highest similarity in the vector space as its source table.

Before calculating TF-IDF on questions and tables, we pre-processed our data to improve the performance. For each question, we transformed all the alphabet characters into lower cases and treated any character other than alphabet characters and numerical characters as separator to split the question into a list of words. We then used Natural Language Toolkit (NLTK) to remove all the stop words in the question and applied stemming to the remaining words. After pre-processing, each question was transformed into a list of word stems. For tables, we iterated through each cell in the table and applied the same pre-processing method. We concatenated the result of each cell in the table so that we got a list of word stems for each table after pre-processing.

Then we calculated TF-IDF of word stems in questions and tables. The inverse document frequency (IDF) of each word stem was calculated on all tables, and the term frequency (TF) of each word stem for each question or table was calculated in the scope of that specific question or table. For each word stem in the questions and tables, we multiplied its TF with IDF to get its TF-IDF value. Then all the questions and tables were mapped to a vector space based on the TF-IDF value of their word stems.

We have explored three different approaches to measure the similarity of the question and the table in the vector space:

- **Cosine similarity of the two vectors**
- **Dot product of the two vectors**
- **Inverse Euclidean distance of the two vectors (after projected to points in multi-dimensional space)**

The results for each approach is presented in Section 8.1

7.1.4 Column Projection in SELECT Clause

Given a question and its source table, in order to build the SQL query, the first task is to determine which columns belong to the SELECT clause of the SQL query. This task can be treated as a binary classification task: given the question and a column in its source table, determine whether this column should be included in the SELECT clause of the SQL query.

The input of our classifier are the following features, which are extracted from a pair of question and column to be processed:

- **The number of columns in the table that the column belongs to**
  The number of columns one table contains influences the probability that any column within the table being included in the SELECT clause significantly. If the table only has one column, then the only column must be included in the SELECT clause to construct the SQL query. As the number of columns in the table increases, the probability of each column being included in the SELECT clause decreases.

- **Proximity of the column contents and the question using word vectors**
  The function of the proximity feature is to measure the similarity of the question and the content of the column. For example, if the question is *Who is the president of the USA?*, the word "president" would be close to "Donald Trump" in word vector space and the column containing "Donald Trump" is exactly the column we want to select. We calculated 4 types of proximity and include them all in this feature. The 4 types of proximity are the average proximity of the column contents and the question, the average proximity of the column contents and the question without stop words, the maximum proximity of the column contents and the question, and the maximum proximity of the column contents and the question without stop words.

- **Column Types**
  In order to identify the correct data type for each column in a table, we built a model for column types recognition. The details of this model is further explained in Section 7.1.6. We use the output after the Softmax layer as our column types features. Our column types feature is a 7-dimensional vector which corresponds to the probability distribution among the 7 column types.

- **Question Types**
  We classified questions based on Li et al.'s work[6]. In this method, we have 6 major types : "ABBREVIATION", "ENTITY", "DESCRIPTION", "HUMAN", "LOCATION" and "NUMERIC". We also included yes or no questions as a seventh major type. In addition, the minor types for "NUMERIC", namely, "date", "count", "period" and "money" were used as we were dealing with lot of factoid questions and needed further granularity. We used the classification API from Madabushi. et.al[7]. This API extends Li et al.'s work [6] and provides usable question type results. We augmented these results by heuristic methods and had 89.2% accuracy on the training set and 96.1% accuracy on the test set.The mapping for question types and column types are intuitive.If the question is asking about the price of some product, then the column included in the SELECT clause should have a "Currency" type. If the question is of type "NUMERIC:date", then we should look for a column with type "DateTime". In our final representation, we have totally 11 different types of questions and our feature of question types is a 11-dimensional one-hot vector.

- **Semantic similarity of the column header and the question**
  The column header might give us important clue of whether this column should belong to the SELECT clause. For example, if the question is *Who is the president of the USA?* and the column header is "president", then there is a high probability that this column should be included in the SELECT clause. Thus we include this feature to measure the similarity of the column header and the question. We applied same pre-processing technique as discussed in section 7.1.3 to the column headers and the question. For each word stem in the column header and each word stem in the question, we calculated the character level edit-distance of the two word stems. Through exploring our dataset, we found that there might be several column headers in the source table all containing the same word from question. For example, there is a question asking about *"What is NAIRU?"*, and the column headers of the first three columns of the source table are "What is NAIRU? CONCEPTS", "History of NAIRU" and "Different concepts of NAIRU". If we only include the minimum edit distance as our feature, then the distance between any of these column headers and the question is equal to zero and the feature is useless. Therefore, to increase the robustness of our feature, we include both the minimum distance and the second minimum distance as our features for all the word stems pairs. So this feature is a 2-dimensional vector that it contains two integer number corresponding to the first and the second minimum distance accordingly.

We concatenated all our features as the input of our model, so the input is a 25-dimensional vector. As shown in Figure 4, Our classifier is a multi-layer perceptron (MLP) with 4 hidden layers of hidden sizes 32, 16, 8, 2 respectively from bottom to the top. We implemented this MLP model in Pytorch[4]. We used SGD and cross-entropy loss to optimize our model. Since the number of positive cases is much smaller than the number of negative cases in our train dataset (Among the total of 2046 columns, 273 are included in SELECT clause, 1773 are not included in SELECT clause), we duplicated the positive cases 6 times to balance our training data.

[4] `https://pytorch.org/`

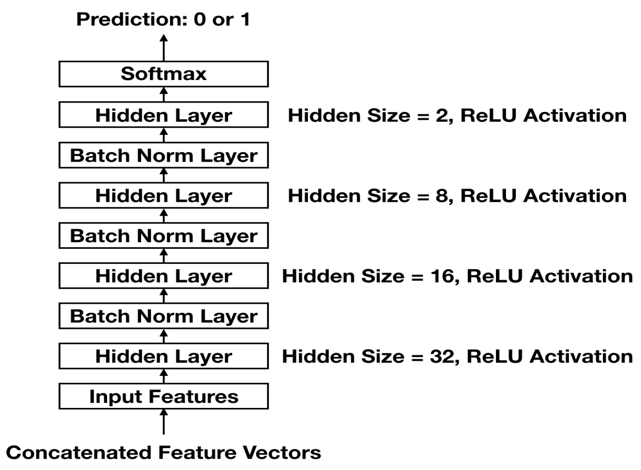

Prediction: 0 or 1
Softmax
Hidden Layer
Hidden Size = 2, ReLU Activation
Batch Norm Layer
Hidden Layer
Hidden Size = 8, ReLU Activation
Batch Norm Layer
Hidden Layer
Hidden Size = 16, ReLU Activation
Batch Norm Layer
Hidden Layer
Hidden Size = 32, ReLU Activation
Input Features
Concatenated Feature Vectors

Figure 4: MLP model for source selection

7.1.5 WHERE Clause Projection

Another critical component of the SQL query is the WHERE clause. Each instance in WHERE clause is a pair of column and key word. Similar to the task of column projection in SELECT clause, we built a binary classifier which takes the question, one column within the table and one word in the question as input. The classifier would output 1 if and only if the input column and the word both appear in the WHERE clause of the SQL query. We did feature engineering on input questions, columns and question words, and then concatenated all the feature vectors and fed into our classifier. The following list all the features we get for this task:

- **Minimum normalized edit-distance between the input question word and the words within the input column**
  For the WHERE clause, the input column exploits the input word to constraint the query result to some specific rows within the table. If the input question word appears in the input column, then the probability of this column and question word pair forming a valid WHERE clause increases.

- **Average character-level cell length with in the input column**
  The SQL query was manually written by students and since it is hard for humans to find key information in extended content, a column with lower average cell length is more likely to appear in the WHERE clause.

- **Number of rows of the input column**
  If the input column only contains a single row, then we may not need to have a WHERE clause in the SQL query. The classifier is this case would have a higher chance to output 0.

- **Whether the input column has already been included in SELECT clause**
  If the input column has already been included in SELECT clause, then it is less likely that it is also used in WHERE clause. We used the ground truth instead of the predicted results for the SELECT clause to build this feature, this was to isolate the prediction errors from the previous model from the current model.

- **Column Types and Question Types**
  We used the same column types feature and question types features as we used in SELECT clause task, since they are still of vital importance for WHERE clause task. In SELECT clause task, for one type of question, the probability that one corresponding type of column to be included in SELECT clause is higher than the remaining types. However, for WHERE clause task, the probability of the column with that specific type appears in the WHERE clause is relatively low. For example, for "When" type question, it is unlikely that we still use a column with "DateTime" type to filter the query result.

- **POS and NER tagging and the dependency parsing of the input word**
  The NLP features such as POS and NER tagging, dependency parsing and so on may give us some clues on the probability of the input word being included in WHERE clause. For POS tagging, we believe some types of words are more likely to be used as keyword in WHERE clause than other types of words. For example, nouns have a higher probability to be selected as the search word in the WHERE clause than adjectives and verbs. Similarly, for NER tagging, entities such as "PERSON" are more likely to be selected as the search word compared to other entities like "DATETIME". Moreover, dependency parsing helps us to identify the root word in a sentence, which has a higher chance to be included in the WHERE clause. Therefore, we included POS and NER tagging and dependency parsing of the input word as features. These feature were converted to one-hot vectors. We list the features we used in Appendix. There are a total of 12 POS features, 6 NER features and 37 dependency parsing features.

We applied the same MLP model used for the SELECT clause task to WHERE clause task, except that the input size was increased from 25 to 77. We also performed up-sampling to balance our training data.

7.1.6 Column Types Recognition

As column type is a very important feature for some sub-problems, we have built a model to recognize different types of columns.

We pre-defined 7 column types, "DateTime", "Currency", "Percentage", "Numerical", "Boolean", "Text" and "URL". We manually labeled all the columns in the train dataset (total of 2046 columns). Our task is that given the content of a column in the table, determine which type this column belongs to. We extracted the following features as the input of our model.

- **The proportion of numerical cells**
  We counted the proportion of numerical cells within the given column. The numerical cell is the cell whose content can be directly converted into a valid numerical value. The function of this feature is to measure the probability of the given column being a Numerical column.

- **The proportion of digit characters**
  We calculated the proportion of digit characters (0-9) with respect to the all characters in the given column. This feature is still to measure the probability of the given column being a Numerical column. If all the characters of the column are digits, then it is very likely that the column is a Numerical type.

- **The proportion of cells containing currency symbols**
  We collected all the currency symbols occurring in our training set and calculated the proportion of cells which contain the currency symbols. This feature is to calculate the probability of a given column in Currency type. The higher this proportion is, the higher probability the given column is a Currency column.

- **The proportion of cells containing percentage symbol**
  Similar as the above feature, we calculated the proportion of

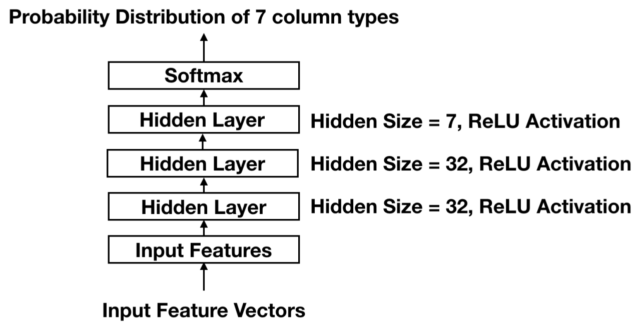

Probability Distribution of 7 column types
Softmax
Hidden Layer
Hidden Size = 7, ReLU Activation
Hidden Layer
Hidden Size = 32, ReLU Activation
Hidden Layer
Hidden Size = 32, ReLU Activation
Input Features
Input Feature Vectors

Figure 5: MLP model for column types recognition

cells which contain the percentage symbol "%". This feature is to calculate the probability of a given column in Percentage type.

- **The proportion of cells containing Boolean values**
  We defined four Boolean values, "yes", "no", "true" and "false". We calculated the proportion of cells which contain the Boolean values. This feature is to calculate the probability of a given column in Boolean type.
- **The proportion of cells containing valid year**
  We defined the integer between 1500 and 2020 as a valid year. We calculate the proportion of cells which contain a valid year. This feature is to calculate the probability of a given column in DateTime type.
- **The proportion of cells containing month values**
  We calculated the proportion of cells which contain month values. Month values are the full name of months, from January to December, as well as their corresponding abbreviations, from Jan to Dec. This feature is to calculate the probability of a given column in DateTime type.
- **The proportion of cells containing weekday values**
  We calculated the proportion of cells which contain weekday values. Weekday values are from Monday to Sunday. This feature is to calculate the probability of a given column in DateTime type.
- **The proportion of cells containing string "http"**
  We calculate the proportion of cells which contain string "http". This feature is to calculate the probability of a given column in URL type.

Give the above features extracted from the input column, we built a multi-layer perceptron (MLP) for column types recognition using Pytorch[5]. The model has 3 hidden layers, with size of 32, 32 and 7 respectively from bottom to the top, followed by a Softmax layer, as shown in Figure 5. The output of the model is a probability distribution among the 7 types given the input column. We assigned the column with the type that has the highest probability and checked with the ground truth data. We have achieved an accuracy of over 93% on our manually labeled training data.

7.2 Evaluation Metrics

7.2.1 Table Type Recognition

We calculated the accuracy of different machine learning models for table type recognition to evaluate their performance on the classification of the input table as a key-value table or an entity-instance table.

[5]`https://pytorch.org/`

7.2.2 Source Selection

For the source selection task, we map questions and tables to a vector space and select a source table based on the similarity of the question and the table in the vector space. We used P@k as the evaluation metric, in which we calculated the precision of the ground truth source table being selected in the top N most similar tables for a given question, with $N = 1, 3, 5, 10$.

7.2.3 SELECT Clause and WHERE Clause

For the SELECT clause and WHERE clause tasks, since we treat each task as a machine learning classification problem, we use confusion matrix to measure the performance of our classifiers. From the confusion matrix, we further calculated the accuracy, recall and precision performance for both train set and test set.

8. EXPERIMENTAL RESULTS

8.1 Source Selection

	Train Dataset			Test Dataset		
	Cosine Similarity	Dot Product	Inverse Euclidean	Cosine Similarity	Dot Product	Inverse Euclidean
P@1	60.5%	68.9%	70.6%	73.4%	71.9%	73.4%
P@3	79.0%	81.5%	81.9%	85.9%	87.5%	87.5%
P@5	84.5%	85.3%	86.1%	90.6%	90.6%	92.2%
P@10	89.9%	91.2%	92.0%	92.2%	95.3%	95.3%

From the table above, we could see that using Inverse Euclidean distance to calculate the similarity gives us the best performance. We achieved an accuracy of 70.6% on train dataset and 73.4% on test dataset. After error analysis on the results, we found that some error cases are not considered as wrong due to the fact that multiple sources could be used to answer the same question. Considering these cases, the adjusted P@1 results using Inverse Euclidean distance method are 76.5% on train dataset and 76.6% on test dataset respectively.

8.2 Table Type Recognition

	Train Dataset	Test Dataset
Logistic Regression	97.9%	100.0%
Decision Tree	98.3%	100.0%
KNN	98.3%	100.0%

We got really promising results on table type recognition subproblem. The best accuracy we achieved is 98.3% on train dataset and 100.0% on test dataset.

8.3 SELECT Clause

	Train Dataset		Test Dataset	
	Predicted:1	Predicted:0	Predicted:1	Predicted:0
Actual:1	238	35	56	10
Actual:0	305	1468	94	399

	Train Dataset	Test Dataset
Accuracy	83.4%	81.4%
Recall	87.2%	84.9%
Precision	43.8%	37.3%

The 2 tables above present the confusion matrix along with accuracy, recall and precision results of predicting the correct columns in the SELECT clause. Our model is able to achieve a good accuracy

(83.4% on train dataset and 81.4% on test dataset) and recall (87.2% on train dataset and 84.9% on test dataset) performance. However, the precision is not very high. We divide the error cases into several different categories and explain in more details in Section 9.2. For this sub-problem, the recall is relatively more important compared to precision, because we want to make sure we include all the targeted columns in our predicted SQL query so that users will not miss any critical information in the returned result. The irrelevant columns appearing in the result just provide users with extra information. However, we still need to further improve our model so that it learns how to get rid of these irrelevant columns.

8.4 WHERE Clause

	Train Dataset		Test Dataset	
	Predicted:1	Predicted:0	Predicted:1	Predicted:0
Actual:1	117	3	18	12
Actual:0	26	5199	36	1731

	Train Dataset	Test Dataset
Accuracy	99.5%	97.3%
Recall	97.5%	60.0%
Precision	81.8%	33.3%

From the 2 tables above, we can see that the prediction accuracy is very high on both the train dataset (99.5%) and the test dataset (97.3%). However, for the recall and precision, the performance is good on the train dataset, but not very good on the test dataset. This tells us that the model is not able to fully duplicate what is learned on the train dataset to the test dataset. This is mainly due to the sparsity of the dataset: not all questions would require a WHERE clause in the converted SQL query. As we can see from the confusion matrix, we only have 120 cases in the train dataset that contain WHERE clause, this does not cover all the different cases for WHERE clause. Hence the model is not able to learn all the general rules to identify the correct "`column` $\sim$ `question_word`"pair in the WHERE clause. We have performed error analysis and divided the error cases into 4 main categories. The details are presented in Section 9.3

9. ERROR ANALYSIS

From the experimental results presented in Section 8, we focus on error analysis for Source Selection, SELECT Clause and WHERE Clause, given the promising result we have achieved on Table Type Recognition.

9.1 Source Selection

When performing error analysis for Source Selection, we noticed that some predictions are not wrong, although the predicted table is different from the actual table as defined in the ground truth. This is due the fact that there are some pairs of tables containing similar information, thus could be used to answer the same question. The reasons why we have similar sources in the dataset are due to three main causes:

- **Case 1: There are some questions that are semantically equivalent.** e.g. *When is Easter this year?* and *What day is Easter on this year?*
- **Case 2: There are some questions that are in the same type and are very close.** e.g. *What is the capital of Louisiana?* and *What is the capital of New Jersey?*
- **Case 3: There are some questions that are asking different information about the same thing.** e.g. *What time does the super bowl start?* and *Where is the Super Bowl being played this year?*

9.2 SELECT Clause

Besides the column-level performance as presented in Section 8.3, we are also interested to learn question-level performance: For a given question, we would like to know if our model has successfully predicted all the targeted columns in the SELECT clause, and if our model has only included the targeted columns in the predicted SELECT clause. We performed this analysis on test dataset: Among all the 64 cases in the test dataset, we have 6 (9.4%) exact matches between the predicted SELECT clause and the actual SELECT clause. Besides, we have another 48 (75.0%) cases that we include all the required columns but also provide additional columns.

We then performed error analysis to study the causes for over-predicted and wrongly-predicted cases. As discussed in Section 7.1.4, two main features used in the model of SELECT clause are question types and column types. Therefore, any mistakes or ambiguity in detecting the correct question types or column types could cause a prediction error in the SELECT clause. We have divided the error cases into six main categories as explained below, all these cases are related to these two types of features to some extent.

- **Case 1: Question type detected wrongly (7 cases)**
  The question classifier API we use is able to classify the questions correctly with 89.2% accuracy on the train set and 96.1% accuracy on the test set. However, there are some cases in which the questions are classified wrongly especially for WHAT type questions. For example, the question *What is Washington Wizards record?* is supported to ask for some records, hence the question type should be identified as "NUMERIC". However, the question classifier detects this question as "ABBREVIATION", therefore, the model would look for "Text" columns instead of "Numeric" columns.

- **Case 2: Column type detected wrongly (3 cases)**
  The MLP model we developed for column type recognition also provides a good accuracy performance of over 93%. But in some cases, due to some special format in the table content, it fails to correctly identify the correct column type which then leads to a wrong prediction in the SELECT clause. For example, for the question *How long do cats live?*, the column "Lifespan" is the targeted column which is supposed to be identified as "Numeric" type. However, as this table only has one row, and the content in "Lifespan" column is *"4-5 years (In the wild)"* which has more alphabets than digits, it is identified as a "Text" type.

- **Case 3: Multiple columns in targeted type (24 cases)**
  The binary classification model we developed for SELECT clause is able to learn the relationship between the question types and column types. However, in some cases, the table has multiple columns of the targeted types. Although we added some semantic similarity features such as edit distance between the column header and the question to help us shortlist the selected columns, it is really hard to get the model distinguish two columns in the same data type and with similar information. For example, for Question *"How many centimeters in an inch?"*, the model selects both "Centimeter" and "Inch" columns in the table. Both these two columns contain numeric data and both column headers appear in the question. Therefore, it is really hard for the model to learn which one should be included in the SELECT clause unless we incorporate more semantic features to identify the "keyword" of the question. Moreover, sometimes we may even

need external information to help us select the correct column. For example, for the question *What is the population of Boston MA?*, the table has two population columns: "2018 Population" and "2016 Population", we would need to know that the current year is 2018 and then get the model to select the correct column.

- **Case 4: Location type detection needed for WHERE-type questions (5 cases)**
One of the limitations in the current column type recognition model is that it is not able to further segregate "Text" type into several sub-classes which includes "Location", "Person", "Description", "Instruction" etc. As a result, we are not able to get a high-accuracy performance for WHERE type questions since we would need to identify "Location" type to answer WHERE type question correctly.

- **Case 5: Human Entity type detection needed for WHO-type questions (2 cases)**
Similarly as what we discussed in Case 4, we would need to identify "Person" type to correctly answer WHO type questions. Right now the model just looks for "Text" type columns, but in most cases, we have multiple "Text" type columns in a table, thus we are not able to achieve a high accuracy performance for WHO type questions.

- **Case 6: Binary questions (4 cases)**
In general, the hypothesis we made for the relationship between question types and column types works well for Wh-type questions such as WHEN, HOW MUCH, HOW MANY, etc. However, for the binary questions where the answers are either "yes" or "no", this hypothesis would not work. We may require more semantic features to get the model fully understand the questions, and then search for the correct columns containing the information to answer the binary questions.

9.3 WHERE Clause

In addition to the column-level performance as shown in Section 8.4, we also checked question-level performance on test dataset: Among the 64 cases in test dataset, we have 34 (53.1%) exact matches between the predicted WHERE clause and the actual WHERE clause. We also have another 8 (12.5%) cases (i.e. Case 1 and Case 2 below) are not considered as wrong because using the predicted WHERE clause also generates the same result as the actual SQL query.

After performing error analysis for WHERE clause, we have identified 4 main categories as explained below.

- **Case 1: Single row tables (7 cases)**
For those tables that have only a single row, the result would be same whether we include a WHERE clause or not in the SQL query. Most of the single row tables are generated after transposing the key-value tables as discussed in Section 7.1.2

- **Case 2: Slight difference on search keyword (1 case)**
Sometimes there might be some slight difference between the predicted search keyword in WHERE clause and the actual one. However, as we introduced a $\sim$ operator as discussed in Section 4.3, it may not affect the final result returned for some cases. For example, for the question *What is Washington wizards record?*, the WHERE clause in the actual SQL query is *Team* $\sim$ *Washington wizards*. Our predicted WHERE clause is *Team* $\sim$ *Washington*. Since in the actual table, the "Team" column only contains "Washington" instead of "Washington wizards", our predicted WHERE clause would return the same result as the one in the actual SQL query, hence is not considered to be wrong.

- **Case 3: External information required (4 cases)**
We also have some cases that would need external information to form the correct SQL query. For example, for the question *Who is the actress that plays Sheldon's mother?*, we would need to have the knowledge that Sheldon's mother is Mary Cooper before we could form the correct WHERE clause which is "`Character LIKE Mary Cooper`".

- **Case 4: Incomplete search keyword due to stop-word removal (3 cases)**
In order to better identify the search keyword in the WHERE clause and clean up the features generated for the model, we made an assumption that the search keyword should not be a stop-word in the question. This assumption helped us to improve the model accuracy, however, we also have few cases predicted wrongly due to stop-word removal. For example, for the question *How many feet are in a mile?*, the expected WHERE clause is "`Mile` $\sim$ `a mile`", but the predicted WHERE clause is "`Mile` $\sim$ `mile`" as stop-word "a" is removed.

10. DISCUSSION

10.1 Summary

The project aims to have a baseline dataset for evaluation of answering natural language questions from structured data. We also aim to find underlying properties of the dataset and create a pipeline that can eventually create an SQL query that generates the structured data for the answer.

The process of generating SQL queries was tackled as solving a series of sub-problems sequentially. We assume that the previous sub-problem has been resolved so that independent success rate of the individual sub-problems can be measured. We provided sub-problems with labeled data and they had a uniformly created train dataset and test dataset.

- **Data Collection**
Table name and column headers were not always available for the tables we collected from the web. In this case, the students collecting the tables were instructed to come up with a table name and/or column headers that was relevant regarding the context (i.e. table, web page). However, the students also knew the question that the table is supposed to answer. The column headers, therefore could have information leaking about the answer that has to be returned for a question. When these questions are in the test set, the gain in accuracy with this leaked information is difficult to gauge. The application scenario assumes that tables are generally extracted independently of the question itself. We can try to solve this by having reliable automated column header generators. However this is a difficult problem to solve.

- **Table Type Recognition**
For classifying the table as an entity-instance type or key-value type, we had the best result with a decision tree classifier. We have achieved 98.4% accuracy on the training set and 100% accuracy on the test set.

- **Table Transpose**
Transposing key-value tables into entity-instance tables provides us with a more homogeneous structure and makes it

easier to run Machine Learning models. Most of the transposed key-value tables have only a single row. However, we also have few transposed key-value tables with multiple rows. For instance, Table 3 is a key-value table with multiple "value" columns. After transpose, Table 4 has multiple rows.

Feature	**1**	**2**	**3**	**4**	**5**
product	acer aspire e15	dell xps 13	hp spectre x360 13	apple macbook pro 13-inch	lenovo ideapad miix 520
lowest price	$349.99	$799.99	$1199.99	$1099.99	$999.99
processor name	intel core i3-7100u	intel core i7-8550u	intel core i7-7500u	intel core i5	intel core i5-8250u
ram	4 gb	8 gb	16 gb	8 gb	8 gb
weight	5.12 lb	2.7 lb	2.83 lb	3.02 lb	2.65 lb
screen size	15.6 inches	13.3 inches	13.3 inches	13.3 inches	12.2 inches

Table 3: Original key-value table with multiple value columns

Product	**lowest price**	**processor name**	**ram**	**weight**	**screen size**
acer aspire e15	$349.99	intel core i3-7100u	4 gb	5.12 lb	15.6 inches
dell xps 13	$799.99	intel core i7-8550u	8 gb	2.7 lb	13.3 inches
hp spectre x360 13	$1199.99	intel core i7-7500u	16 gb	2.83 lb	13.3 inches
apple macbook pro 13-inch	$1099.99	intel core i5	8 gb	3.02 lb	13.3 inches
lenovo ideapad miix 520	$999.99	intel core i5-8250u	8 gb	2.65 lb	12.2 inches

Table 4: Transposed key-value table with multiple rows

- **Source Selection**
  The source selection problem worked best with calculation of inverse Euclidean distance on pre-processed question and table data. It achieved a result of 76.5% on the train dataset and 76.6% on the test dataset after considering semantic similarity of the proposed table and the actual table. Since we work with similarities based on bag-of-words model for the questions and tables, contextual information would get lost. Most errors were from this loss of context. Namely, closely related questions where only one word is changed, would typically get misclassified. However, this approach still works better than a learning model as we have limited training data and word distribution similarities is still a reasonable approximation for mapping tables to questions. We should note that as we add more new sources, it is possible that the performance might drop due to confusion between data sources.

- **SELECT Clause**
  We used a Multi-layer perceptron neural model for classification in the SELECT clause problem as well as the WHERE clause problem. We had a precision of 43.4% and 37.3% respectively for train and test set, which corresponds to the highest scoring result. When considering recall, which accounts for the presence of the correct results returned by the classifier, we had 87.2% and 84.9% respectively. This is a good result as it would provide an opportunity to rerank the results using additional features.

  Errors for the SELECT clause problem arose from multiple sources. Misclassification of the question type and the column type led to some. Column type recognition may be improved by using pre-trained models or clustering techniques. There needs to be higher granularity for column type recognition such as those identifying a location or a human entity. Ambiguities that resulted from a table having multiple columns of the same type also had an effect on the accuracy. Yes or no questions were also difficult to get right possibly because it required semantic inferences.

- **WHERE Clause**
  For the WHERE clause problem, the precision is more important and it we had 81.8% and 33.3% for train and test sets respectively. Due to the data sparsity issue in our current dataset, we are not able to fully extend what is learnt on train dataset to test dataset.

  Some errors in misclassification of (column, question word) pairs for the WHERE clause problem were on tables that had a single row. This would not be an error in the final result pipeline. We use a $\sim$ operator to identify the column that most closely resembles the word in the question we need to consider. Therefore small errors in the selection of the question word will also not be a error in the final result. We pre-process the question to remove stop words from consideration. However, in some cases, the final result required their use. This is would be a hard problem to solve, as we have observed that accuracy drops when stop words are included. Additionally, questions requiring semantic inferences were a bottleneck here as well.

10.2 Complex cases and Edge cases

We have proven that our approach works for most general cases. However, there are some complex cases or edge cases need to be further studied after we collect more data.

10.2.1 AND/OR operator

For some questions, we may need to put AND or OR operators in the converted SQL query. For example, for Question *"What year was Brock Lesnar's last UFC fight?"*, the ground truth SQL query is `SELECT "Date" FROM "Brock Lesnar_Tabe" WHERE (("Fighter_1" ~ "Brock Lesnar") OR ("Fighter_2" ~ "Brock Lesnar")) AND "Event_Name_1" ~ "UFC" ORDER BY "Date" DESCENDING LIMIT1`

10.2.2 Sub query

For some questions, we would need to use sub-query to generate a complete SQL query to locate the correct answer. For example, for Question *"Who became president after John Kennedy?"*, the ground truth SQL query is `SELECT "President" FROM "List of Presidents of the United States" WHERE "Number" > (SELECT "Number" FROM "List of Presidents of the United States" WHERE "President" ~ "John Kennedy") ORDER BY "Number" ASCENDING LIMIT 1`

10.2.3 Aggregate function

For some questions, we may not be able to find a direct answer in the table. However, we could get the answer using aggregation functions in SQL. For example, for question *"How many science jobs* *in Rochester NY?"*, the ground truth SQL query is `SELECT COUNT (Title) FROM "Table_1" WHERE "Location" ~ "Rochester NY"`

10.2.4 JOIN operator

As discussed in Section 9.3, for some questions, we may need external information to form the correct SQL query. If there is another data source containing this external information, we could use a JOIN operator to combine information from two sources to solve this problem.

10.2.5 Questions with answers change over time

We also have questions that have ambiguities due to under specification such that the answers change with time. For example, for Question *who won the super bowl?*, the answer would be changed by year. Assuming that the table would be updated so that it contains the most recent record, we have two ways to solve this problem to make sure we always get the most updated answer.

- We could sort the table by year and get the most recent one. i.e. `ORDER BY year DESCENDING LIMIT 1`.
- We could call an external function which returns the current year from the environment. i.e. `WHERE year = EXTERNAL ("current year")`.

This latter approach also applies to questions that require users' location. For example, *"What is the best restaurant near me?"*.

11. LESSONS LEARNED AND REFLECTIONS

- **Semantic inference problems are hard to solve**
  This is a common problem in the question answering domain. This pertains to questions that require higher order reasoning or external knowledge. We saw examples of both kinds of questions in 9.3.
- **Data set needs to be bigger**
  We worked with a data set that was only 302 tables which were not homogeneous enough in terms of domain or complexity. Machine learning models may not be able to learn all the general patterns and relationships from the features.
- **Data analysis**
  Close analysis of individual tables provided insights that were proved to be useful. The decision to transpose the key-value tables was taken after we discovered and defined a few classification rules from the initial data analysis. Similar close scrutiny for the SELECT clause and WHERE clause problems revealed limitations of our approach.
- **Data distribution**
  The train dataset and test dataset do not have a similar distribution in terms of question type or complexity. This means that heuristic approaches that worked in the training set may not be fully extended into the test set.

12. FUTURE WORK

- **Expand the data set**
  We need to add more questions and tables to the data set. We have observed that in addition to enabling effective use of machine learning techniques, we infer valuable insights from individual tables themselves. Moreover, we also need to have enough training examples to identify the correct features and models to be used for ORDER BY and LIMIT clause problems.
- **Build the complete pipeline**
  After we complete all the sub-problems, we need to integrate the individual components in sub-problems. We need to use the output of our own model instead of the ground truth to build features that require results of the previous sub-problems. This enables us to define global error rates and use evaluation metrics such as F1 to measure the performance of the system. This would also enable us to identify performance bottlenecks in a global context.
- **Learn to identify more column types**
  As discussed in Section 9.2, some question types such as WHERE and WHO have a lower accuracy performance compared to other question types, this is because we are not able to provide a higher granularity result on column type recognition. We need to study how to break down "Text" columns into several different types such as "Location", "Person", "Description", "Instruction", etc.
- **Incorporate more semantic features**
  Although semantic inference problems are not easy to solve, from our error analysis, we know that we are able to further improve the system performance by incorporating more semantic features. This could help us to better answer the binary questions, and also questions like *How many centimeters in an inch?*
- **Explore the use of pre-trained models**
  An alternative or addition to expanding the data set is using pre-trained models such as those used in [9].
- **Re-ranking the top results**
  When we have multiple sources that might answer a question, we should evaluate the top k sources and see if we get the best answer. We could also explore if a composition of re-ranked results could generate a better answer.

13. CONCLUSION

We proposed to understand the problem of question-answering from structured sources by forming a dataset for modelling and providing well-defined sub problems. We believe that we can compose the results and provide a baseline performance for solutions to this problem. In this work, we were able to provide the division for sub-problems and implement baseline models for each sub-problem. We also performed error analysis on each model.

The table type recognition problem was solved using a decision tree classifier that relied on table level features such as number of columns, variation of cell length and so on. The source selection problems was solved with IR techniques. Both solutions were reasonably accurate and further performance improvements may be gained with fine-tuning.

The SELECT clause problem can be seen as a standard question answering problem that involves mapping question types and column types effectively. We observed limitations with type classification. We also observed problems with disambiguation with multiple matches and semantic inference problems.

The WHERE clause problems proved the hardest as question type knowledge did not have a direct effect on the results. Further, this needed both column and the question word to be guessed correctly and therefore the entropy for the WHERE clause is higher.

Our approach by dividing the problem into several well-defined sub-problems makes it easier to identify the error causes and interpret required areas of focus. After collecting more data and building models for ORDER BY and LIMIT clauses, we would like to complete the framework integration.

14. REFERENCES

[1] R. Bordawekar and O. Shmueli. Enabling cognitive intelligence queries in relational databases using low-dimensional word embeddings. *arXiv preprint arXiv:1603.07185*, 2016.

[2] J. Cheng, S. Reddy, V. Saraswat, and M. Lapata. Learning structured natural language representations for semantic parsing. *arXiv preprint arXiv:1704.08387*, 2017.

[3] S. Gardiner, A. Tomasic, and J. Zimmerman. Smartwrap: seeing datasets with the crowd's eyes. In *Proceedings of the 12th Web for All Conference*, page 3. ACM, 2015.

[4] S. A. Hayati, R. Olivier, P. Avvaru, P. Yin, A. Tomasic, and G. Neubig. Retrieval-based neural code generation. *arXiv preprint arXiv:1808.10025*, 2018.

[5] S. K. Jauhar, P. Turney, and E. Hovy. Tables as semi-structured knowledge for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 474–483, 2016.

[6] X. Li and D. Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[7] H. T. Madabushi and M. Lee. High accuracy rule-based question classification using question syntax and semantics. In *COLING*, 2016.

[8] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

[9] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

APPENDIX

A. POS TAGS FEATURE LIST

Type	Description
ADV	adverb
VERB	verb
DET	determiner
ADJ	adjective
PROPN	pronoun
ADP	adposition
PART	particle
NOUN	noun
NUM	numeral
PRON	pronoun
INTJ	interjection
CCONJ	coordinating conjunction

B. NER TAGS FEATURE LIST

Type	Description
PERSON	People
LOCATION	Location
DATETIME	Date and Time
QUANTITY	Measurements, as of weight or distance etc.
ORGANISATION	Companies, agencies, institutions, etc.
NONE	None of above

C. DEPENDENCY PARSER FEATURE LIST

Type	Description
acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list
mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement

More detailed description is in Universal Dependencies website[6].

[6] `http://universaldependencies.org/u/dep/all.html`