Translating Natural Language Queries to SPARQL

A Project Report

Presented to

Professor Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements of the Class

CS 297

By

Shreya Satish Bhajikhaye

December 2020
Abstract

RDF is linked data represented as a graph. The semantic web uses this framework to collate all information on the web. SPARQL language was created to query data in a RDF model. There has been a lot of research done to improve the ease of use of RDF data for searching and querying purposes. This project focuses on creating a RDF-based question answering system. The input for the system is a query statement in English, which is translated to the equivalent SPARQL query. Upon reading a complete sentence, the program identifies the words as either subject, object or predicate and builds the respective query clauses.

As part of CS 297, preliminary analysis for the project was completed. It includes understanding the SPARQL language and its usage. The analysis explores existing question answering systems and the related neural network models. Finally, it tries a proof of concept to use Sequence-to-Sequence learning to parse the sentence and identify the entities that will form the SPARQL query.

Keywords – RDF, SPARQL, Question Answering System
# Table of Contents

I. Introduction ........................................................................................................................................ 4  
II. RDF and SPARQL ............................................................................................................................. 5  
III. Word and Character Level Embedding ........................................................................................... 9  
IV. Named Entity Recognition ............................................................................................................... 13  
V. Entity Matching .............................................................................................................................. 15  
VI. Conclusion ..................................................................................................................................... 18
I. INTRODUCTION

The semantic web is all about creating a network so that machine-controlled devices can read all information on it. In this growing digital era, it aims to convert the unstructured and semi-structured documents on the World Wide Web into a standard format that computers can process easily. RDF is the foundation of the Semantic Web and gives it flexibility. The entire meta data for the semantic web has been constructed using RDF framework. RDF data is represented as a graph and is used to build relationships between entities and properties called knowledge graphs. These linked data are very useful in search and query processes because of the inbuilt relations in them. This is one of the reasons that many question answering systems have an underlying RDF database. SPARQL is a standard language developed to query graph databases represented as RDF triples. This project is to create a Question Answering system that can translate queries in English to SPARQL.

With the development of question answering systems, it has become easier for end users to access knowledge bases and get concise and accurate results. Many state-of-the-art methods can be applied to generate SPARQL queries from question answering system. It is important to generate these queries automatically because SPARQL is a professional level language that can be difficult to learn for most users. Song, Huang and Sun [1] make use of query expansion techniques and semantic query graph generation to map the subgraphs to a logical form. Dai, Li and Xu [2] proposed a deep recurrent neural network along with conditional probabilities based on neural embedding to answer single fact questions. Liu et. al [3] described a model that ranks the subject-predicate pair to retrieve relevant facts from the question. It solves the problem of out-of-vocabulary words and disambiguation with word and character level embedding. The
method to generate and evaluate SPARQL query from natural language question (NLQ) is mostly an open research problem. The approach proposed in this report starts with a linguistic analysis of the question with part-of-speech tagging. Instead of using the general Named Entity classifiers, the encoder-decoder model classifies them as either subject, object or verb. To match these entities, we search through the RDF database for an item with the same name in part or full. It also replicates the character-embedding model from an existing system to overcome the challenge of out-of-vocabulary words.

The rest of this paper is organized as follows: Section II highlights the concepts of RDF data, SPARQL language and using the Wikidata database. Section III studies an existing Question Answering system in-depth and replicates a part of it. Section IV describes the idea of part-of-speech tagging using neural networks. Section V completes the experiments by extracting the entities of a sentence from the Wikidata database. Lastly, section IV concludes the paper with the predicted scope for implementation of a complete framework in CS298.

II. RDF AND SPARQL

The pre-requisite to start the study was a good understanding of RDF data and using SPARQL to query it. The objective was to write at least five complex queries that provided an understanding on the kind of query statements that can be built with SPARQL and the correct syntax of each clause.
RDF data model contains a basic unit of information called as a triple. The triple is a combination of the subject, predicate and object of a statement. More technically, it is referred to as the resource identifier, an attribute or property name and an attribute or property value. These triples can be stored in files with different serializations like N3, XML or Turtle. RDF is useful for implementing graph-oriented databases with complex systems of data. Unlike relational databases, this data model does not require the user to know the structure of the data. Querying with SPARQL becomes easier with limited schema knowledge. Wikidata is one of the popular databases that can be queried over the web.

Below are five queries that were executed on Wikidata and their results:

1. Bubble chart display of the total number of wins by a country at the 2016 Rio Olympics

```sparql
##defaultView:BubbleChart
SELECT ?country ?countryLabel (COUNT(?m_event) AS ?count)
WHERE
{
  wd:Q8613 wdt:P527 ?event.
  ?event wdt:P527 ?m_event.
  ?m_event wdt:P1346 ?person.
  OPTIONAL {?person wdt:P1532 ?country.}
  OPTIONAL {?person wdt:P17 ?country.}
  OPTIONAL {?person wdt:P27 ?country.}
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" }
}
GROUP BY ?country ?countryLabel
ORDER BY DESC(?count)
```

![Figure 1. Bubble chart with the share of wins of each country at the 2016 Rio Olympics](image)
2. Total population in the Bay Area

```sparql
SELECT distinct ?area ?areaLabel (SUM(?popn) AS ?total_popn)
WHERE {
    ?item wdt:P361 wd:Q213205.
    ?area wdt:P31 wd:Q1907114.
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
GROUP BY ?area ?areaLabel
```

"Figure 2. Total population in all metro cities in the Bay Area"

3. Lexemes that mean apple in different language

```sparql
WHERE {
    ?l a ontolex:LexicalEntry ;
    ontolex:sense ?sense ;
    dct:language ?language ;
    wikibase:lemma ?lemma.
    ?sense wdt:P5137 wd:Q89 .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
ORDER BY (LCASE(?languageLabel))
```

"Figure 3. List of lexemes in Wikidata that refer to apple in different languages"
4. Places within 1km distance of San Jose State University

```
SELECT ?place ?placeLabel ?instanceLabel ?dist
WHERE
{
  wd:Q498526 wdt:P625 ?loc .
  SERVICE wikibase:around {
    ?place wdt:P625 ?location .
    bd:serviceParam wikibase:center ?loc .
    bd:serviceParam wikibase:radius "1" .
  }
  OPTIONAL { ?place wdt:P31 ?instance }
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" }
  BIND(geof:distance(?loc, ?location) AS ?dist)
}
ORDER BY ?dist
```

![Figure 4. Locations within 1km of San Jose State University](image)

5. List the count of translations for a disease from the disease ontology

```
WHERE
{
  ?disease wdt:P699 ?doid ;
  rdfs:label ?label ;
  rdfs:label ?enLabel .
  FILTER (lang(?enLabel) = "en")
  BIND (lang(?label) AS ?language)
}
GROUP BY ?disease ?doid ?enLabel
ORDER BY desc(?languages)
```
III. WORD AND CHARACTER LEVEL EMBEDDING

From reading surveys and conference papers, it could be seen that both rule based and machine-learning techniques can be applied to build question-answering systems over knowledge base. The area of interest for this project was the related work that concentrated on using machine-learning networks to learn the question structure and build the equivalent SPARQL query.

Lin et al.[3] described an unsupervised approach to train the model to make all decisions. It learns the relevant facts from the question to retrieve a ranked list of subject-predicate pairs. The network contains a nested word and character level encoder, which handles out-of-vocabulary words and exploits word semantics. This was achieved with the word representation model as shown below.
The word representation encoder consists of a word-embedding module and a character level embedding module. Pre-trained Glove embedding [4] are used to create the word vector part of the representation. For every word that was discovered, custom character embedding are created to feed into an RNN that result in the final character vector. The word and character vector are combined as the word representation for a candidate subject or predicate.

This word representation is an important part of the system defined in [3]. The user enters a question that is separated into a list of facts. All words in this fact list are encoded as their word representation using the model described below. According to [3], the cosine similarity between the word representation of the fact list and the word representation of candidate subject-predicate pairs ranks the possible results. The word level embedding is defined as the dot product of the Glove embedding and the one hot vector representation of the word in the input sentence.
The character-level embedding model observes characters in a word for a predefined window size. For all context characters, the length of the list is the window size defined. For the word ‘hainan’, with a window size = 2 the character contexts are defined as –

```
target: [h]  context: [a i <PAD> <PAD>]
target: [a]  context: [h i n <PAD>]
target: [i]  context: [h a n a]
target: [n]  context: [a i a n]
target: [a]  context: [i n n <PAD>]
target: [n]  context: [n a <PAD> <PAD>]
```

```python
def getWordEmb(words):
    # loading GloVe embeddings to embeddings_index
    path_to_glove_file = "/content/drive/My Drive/Masters Project/Datasets/glove.6B.100d.txt"
    embedding_index = getWordEmbedding(path_to_glove_file)

    word_index, word_encoded = encode_tokens(words)

    embedding_matrix = createEmbeddingMatrix(word_index, embedding_index)

    # Word embeddings as a dot product of the Glove embedding and the one-hot vector
    word_embedding = np.dot(word_encoded, embedding_matrix)

    return word_index, word_embedding
```

`Figure 7. Code segment that defines the word embedding model`
Depending on the character vocabulary, every character is encoded as a one hot vector. This model recognizes a total of 37 characters – the padding symbol [PAD], lowercase alphabets [a-z] and numbers [0-9]. Based on the characters before and after, the LSTM model learns the target character. The embedding layer that connects to it has the character embedding as its final internal states.

The pre-trained word embedding used are Glove embedding from the Stanford NLP publications [4]. The embedding file has 400K words with each having 100 dimensions. A subset of 50,000 words from the Brown Corpus Dataset [5] was used to train the character embedding for all alphabets and numbers.

Setting epoch count = 100 and embedding dimension = 100.
For window size = 2, the accuracy of the character embedding model is 66.90%.
For window size = 3, the accuracy of the character embedding model is 74.90%.
Evaluating for input question – “What cyclone affected Hainan?”
The recognized facts are [ "cyclone", "affected", "hainan" ]
Word index: {'cyclone': 1, 'affected': 2, 'hainan': 3}
One hot encoded vector for the question:   

\[
\begin{bmatrix}
1.0.0. \\
0.1.0. \\
0.0.1.
\end{bmatrix}
\]

The character embedding for the word 'cyclone' are of shape [7, 100], for 'affected' are [8, 100] and 'hainan' are [6, 100]. When these are input to a single-layer GRU, the resultant character vectors are [1,100] for every word. The final word representation are a combination of the word vector [1,100] and the character vector [1,100].

IV. NAMED ENTITY RECOGNITION

To implement the entity identification aspect of the project, a neural network model based on sequence-to-sequence learning is proposed. It looks at the sentence structure from a subject and predicate perspective. It identifies the word in a sentence as either a subject, verb or an object. These are the basic entities required to construct the clauses of the SPARQL query.

The model structure is an encoder-decoder model. Usually sequence-to-sequence learning is used to translate between two natural languages. It converts sequences from one domain (here, English sentences) to another domain (the subject, verb and object mapping). The translation is completely at word level here. The algorithm starts with an input sentence in English and a target sentence with every word identified as either a subject, verb or an object.

There are two LSTM networks in the algorithm. The encoder LSTM takes the input sequence of words and processes it for every time step, learning the information for that word. The final output of the encoder is discarded and the final states of the LSTM cell are used as intermediate vectors. The final states of the encoder are initialized as the initial states of the decoder LSTM cell. The decoder takes input as the start marker along with the required output.
At each time step, the LSTM trains on a word to learn the next word in the sequence. The target sequence for a decoder is constructed as the stop marker with the desired output.

The first dataset was created with 20 random sentences with an average length of 15 words. The words were mapped to the respective entity as either S, V or O in a separate target.txt file. The second training and testing dataset consisted of 3000 sentences with an average length of 25 words. These sentences were extracted from the Wall Street Journal articles in the Penn Treebank corpus. A similar mapping of S, V and O was done in a separate target file for the sequences. Training the 20-sentence dataset for 40 epoch results in an accuracy of 95%. On further examination, it was inferred that these results are a cause of the extra padding added at the end of each sentence. For shorter sentences, the padded characters are more than the actual informative characters of the input sentence. This indirectly increases the training accuracy of the model.

The results for the test dataset with 3 sentences were –

```
Input sentence: they dont get along together
Actual tags: START_ s v v v v _END
Predicted tags: s v v

Input sentence: do you like your boss
Actual tags: START_ v s v o o _END
Predicted tags: s v v o

Input sentence: what are you listening to
Actual tags: START_ o v s v v _END
Predicted tags: s v v o
```

Figure 10. Results of entity classification for a small test dataset
For the 3000 sentence dataset, the results looked like –

![Image showing results of entity classification after training on a larger dataset]

There are a few issues with the results, the most prominent one being the length of the predicted tags. The model does not recognize the _END marker and the padding character correctly. To improve on this a custom loss can be defined to penalize the model over going over the input length. Another issue is the model always recognizes the start of the sentence as a subject token. This could be because of insufficiency of data and variety in it. The dataset needs to be improved to have sentences of all length with varied contexts.

V. ENTITY MATCHING

Wikidata is an easily accessible RDF database over the web. Every resource in it is classified as either an item or property. Usually the subjects and objects are represented as the item and the predicate are represented as property. Once the parts of a triple are identified by methods like the previous POS tagging, their respective identifiers need to be fetched. This is required because unlike SQL queries, SPARQL does not use resource names but rather resource
identifiers. In the select and where clauses we refer to the resource as either Qxx for an item with identifier number xx or Pxx for a property with the identifier number as xx.

To extract the identifiers for the subject, predicate and objects in the input query we wrote a program to connect to the dump and find all relevant items. The first step was to try to download the Wikidata dump on local disk but due to a size of ~56GB it wasn’t possible. Instead, the file could be connected and unzipped batch wise in memory to avoid downloading it. The data format was that of an array of JSON objects. Each JSON object had multiple fields that described a resource. The relevant fields for this study were type, id and label. The type was defined as either an item or property. Other fields included were aliases, description and claims. The labels also included names of the resource in other languages.

```
{
  "id": "Q99",
  "type": "item",
  "labels": {},
  "descriptions": {},
  "aliases": {},
  "claims": {},
  "sitePaths": [],
  "lastrevid": "19202163",
  "modified": "2015-02-10T12:42:02Z"
}
```

Figure 12. JSON format of an item/property in the Wikidata dump

```
"labels": {
  "en": {
    "language": "en",
    "value": "New York City"
  },
  "it": {
    "language": "it",
    "value": "Città degli Stati Uniti d'America"
  }},
  "descriptions": {
    "en": {
      "language": "en",
      "value": "Largest city in New York and the United States of America"
    },
    "it": {
      "language": "it",
      "value": "Città degli Stati Uniti d'America"
    }
  }
```

Figure 13. Format of the label field in the JSON object

The experiment consumed 5 million JSON objects, of which 1.2 million were discarded for not having a label in the English language. The results for 3 kinds of input queries are shown:

```
[ ] Input = "What cyclone affected Hainan" search(input.lower())

what : Q22421113
cyclone : Q7699842
word not in dict : affected
word not in dict : hainan
```

Figure 14. Search results for the query: What cyclone affected Hainan
In the last two queries, the terms ‘World War II’ and ‘San Jose’ were searched as separate words. To make sure that the program also takes into consideration such situations, the next part searches for all consecutive segments that exists in the database.

The method is very primitive but works for short sentences since the iterations would be limited. For longer sentences, a more efficient technique would be to search based on character-grams of the words.
VI. CONCLUSION

In this semester, we experimented with the entity recognition and entity matching aspect of the project. For classifying the entities as subject, object or verb we tried an embedding based model as well as the sequence-to-sequence learning model. This experiment helped in realizing that semantic analysis on the input query was possible without using embedding. It also helped to recognize the kind of variation needed in the test dataset. For entity matching, we used Wikidata as the knowledge base in which all entities were searched. In this experiment, we buffered a large dataset and parsed every object in it. We discovered the challenges of matching phrases in the database that needs to be improved. In addition, a need to distinguish between words as items or property was found.

In the next semester, the focus is on combining the elements created and improving the test dataset. We would capture baseline results and improve the learning by adding module for word sense disambiguation. The identified subjects and predicates then will be combined into different clauses to form the SPARQL query.
REFERENCES


[5] https://www1.essex.ac.uk/linguistics/external/clmt/w3c/corpus_ling/content/corpora/list/private/brown/brown.html