**CS 6320 Project Report**

**Question Answering System**

**Group: Grace & Manpreet**

**Group Member:**

**pxy200000 Pin-Hsuan, Yao**

**mxs200009 Sandhu Manpreet**

**Problem Description**

The aim is to build a question-answering system that uses NLP features and techniques which can extract information from the provided SQuAD dataset and return the sentences containing the answer for the input question.

**Proposed Solution**

We use NLP techniques to extract the sematic & syntactic information from these articles and use them to find closest answer to the user’s question. We’ll extract NLP features like tokenized words, POS tags, lemmas, synonyms, hypernyms, meronyms, holonyms, dependency tree, name-entities etc. for every sentence, and use SOLR tool to index these extracted features and sentence. We’ll extract the same features from the question and form a SOLR search query. The questions can be subcategorized into Who, What and When questions. Based on the subcategory of the question and the query string generated from the question, we return ten sentences with their features and their relevant score. Based on their score, we first smooth it and then calculate how many features do they match with the question. Finally, we pick the sentence with the highest score.

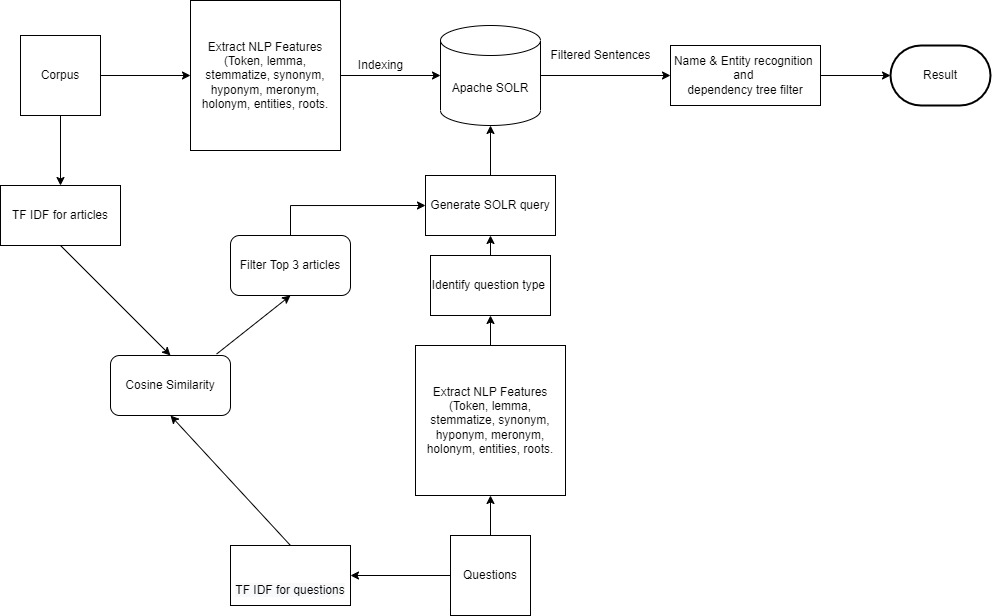
**Implementation Details**

**Programming Tools**

* Python 3.8.5
* Spacy library (name entity recognition, dependency tree)
* Apache Solr 8.10.1

Run Solr and create the collection named “gettingstarted”

* NLTK library (wordnet, stopwords, PorterStemmer, Lemmatizer, POS tags)
* Scikit-Learn ( TfidfVectorizer, cosine\_similarity)

**Architectural Diagram**

*Figure 1 Architectural Diagram*

**Step 1:**

* Read the file and do sentence tokenization.
* Extract the NLP features from each sentence including, word tokenization without stop words, lemmatization, stemming, synonyms, hypernyms, holonyms, meronyms, named entity, dependency parsed tree.

**Step 2:**

* Connect to Solr collection which named “gettingstarted”
* Send the sentence with the filename and it’s NLP features to Solr for indexing.
* 一張含有 桌 的圖片

  自動產生的描述For indexing, we send a sentence-based dictionary in to Solr.

*Figure 2 Index structure in Solr*

In above figure 2, because the data is too long, I use JSON viewer to format it. You can easily see the structure from this figure.

**Step 3:**

* Read the questions and extract NLP features from questions, including tokenization, lemmatization, stemming, wordnet, name entity recognition, dependency tree.
* Using TF-IDF to get top-3 relevant articles with documents and given question.

**Step 4:**

* Identify “what” or “when” or “who” question.
* Form the query based on features for each type of question.
* We use tokens, 3 relevant articles, lemma, stemma, entity text, entity labels, synonyms to form the query.
* For entity labels, we use “DATE” or “TIME” for “when” question, “ORG” or “PERSON” for “who” question.

Figure 3 is the example of forming the query. The question is “What was the capital of the Safavid Dynasty?”

一張含有 文字 的圖片

自動產生的描述

*Figure 3 Query Example*

一張含有 文字 的圖片

自動產生的描述

*Figure 4 Search Result*

**Step 5:**

* Get scores from the query result and form a scoring system based on their sentence features.
* In this scoring system, we use dependency tree features of question to identify the direct object and subject, whether these texts exist in the sentences or not. If they exist, we add score to their original score.
* Then, we use name entity recognition to match sentence with question. We first identify whether there is entity label for each type of question, such as “DATE” or “Time” for When, “ORG” or “PERSON” for Who. If the sentence has that kind of label, we will first give them reward then match their entity text, see whether it appears in the question, if yes, then reward the sentence again. On the other hand, if there is no such kind of label, we then see their entity text, whether it appears in the question. If yes, then we give that sentence reward.
* Next, we look into token, if it appears in the question’s synonym, then we give the sentence reward, else if it appears in the question’s hypernym, we give the sentence reward, else if it appears in the question’s hyponym, we give the sentence reward.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Entity\_text | Token | Entity\_label | Lemma & Stem | Synonym | Hypernym | Hyponym |
| reward | 5 | 10 | 5 | 5 | 5 | 1 | 1 |

* After calculating, we get the best answer with the top score and store it in the list named “best” as the format [Article\_Name, Question, Answer].

*一張含有 文字 的圖片

自動產生的描述Figure 5 Score System example*

Figure 5 is the example of score system, the search question is “Who mediated the truce with Khomeini?”. You can see that the system selects the best sentence with the highest score.

**Step 6:**

* Getting the best list and output as an csv file.

**Result & Error Analysis**

Figure 6 shows our result of Sample Question Answering Data.

一張含有 文字, 文件, 螢幕擷取畫面 的圖片

自動產生的描述The accuracy of this data runs quite well.

*Figure 6 Result of Sample Question Answering Data*

We have noticed that it might select the wrong answer when sentences having same entity, such as the question “What is FARMS” from 281.txt, our answer for that question is “Thus was born the FARMS Critical Text Project which published the first volume of the 3-volume Book of Mormon Critical Text in 1984.” The answer is “By 1979, with the establishment of the Foundation for Ancient Research and Mormon Studies (FARMS) as a California non-profit research institution, an effort led by Robert F. Smith began to take full account of Larson’s work and to publish a Critical Text of the Book of Mormon.” They both have FARMS, but in our search result, our answer has higher score than the correct answer.

In figure 6, you can see that the system selects the wrong answer in row 21, because it also has the Supreme Leader.

**Problems Encountered**

1. **Not finding relevant articles in Solr search query.**

When we first use TF-IDF to find the top 3 relevant articles, the accuracy of finding the correct answer was lower than not using TF-IDF. We investigated the search result, and we discovered that the result contains articles which is not in the set of relevant articles. It turned out that it is because of q.op which is the query operator. In default, q.op is OR and that’s why we had sentences out of the relevant articles. We set q.op to AND via pysolr search.

1. **Search result is hard to discriminate.**

After searching, we tried to get the score of each sentence by only counting their features, however, this is hard to discriminate their score. Thus, we tried to use the relevance score from Solr given the weight that we set in query. However, some sentences score is too large, and it is not enough to say it is the best sentence via solr search. In this case, we smoothed the score. We divided the score by 4 and using this score to be the base score then we calculate the features. In the below figure, you can see that we have 2 sentences with the same score, but it only chooses the first sentence.

一張含有 文字 的圖片

自動產生的描述

*Figure 7 Search result hard to discriminate*

1. **Some punctuation and useless words still exist in the sentence.**

Although we extracted the stop words while tokenizing, there still some useless words or punctuation exists. For example, ``’’, they still exist after tokenizing. Therefore, we had to make sure it will not be counted when calculating the features.

一張含有 文字 的圖片

自動產生的描述

*Figure 8 Check if the token is not in ``’’*

1. **TF-IDF is a case sensitive model.**

Previously, we used lowercase to transform the model, however, we ignored the fact that articles may use the abbreviation such as FARMS, which is stands for Foundation for Ancient Research and Mormon Studies. In this case, if we ignore the case, TF-IDF will match the articles which contain farms, which means an area of land that is used for growing crops and raising animals. Therefore, we then use the original cases to transform and fit the model.

1. **Some word stays the same after lemmatizing.**

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. However, it is hard to be searched in synonym with only lemmatization. Also, the past tense may not include in question. Therefore, we introduce Porter Stemmer in NLTK.

For example, called in lemma is still “called”, but in stemming is “call”

一張含有 文字 的圖片

自動產生的描述

*Figure 9 example of the lemmatization and stemming difference*

**Pending Issues**

1. **spaCy coreference tool seems weird after sentence tokenization.**

一張含有 文字 的圖片

自動產生的描述We previously wanted to use spaCy coreference in our project. Although we finish doing the coreference part. It became weird when we did sentence tokenization. We tried using different sentence tokenizing tool. It is still weird. Therefore, we gave up using coreference as a tool. If we have enough time, we can implement sentence tokenization on our own and maybe this will work for coreference.

*Figure 10 Coreference Issues*

1. **The accuracy still needs to be improved even using dependency tree.**

The accuracy still needs to be improved.

**Potential Improvements**

1. **Using more dependency features to get the correct answer.**

In our code, we only use dobj and nsubj to check whether sentences’ structure match to the question. We can use more dependency tree’s feature to get the correct answer.

1. **Using Coreference as a feature to extract the correct sentence.**

Just like we mentioned previously, because we failed at using it, if we have more time, we want to implement Coreference in our code and make use of it.

1. **Using Elastic Search instead of Apache Solr.**

It seems that Elastic Search contain more features and tools than Solr. It will be more convenient to use Elastic Search than Apache Solr.