Project Report

Spring 2019

Question Answering System

CS 6320: Natural Language Processing

Submitted by:

Team: Mustang

Members:

- Santhosh Medide (sxm174930)
- Aashaar Panchalan (adp170630)

Problem Description

Today the world is full of articles on large variety of topics. We aim to build a question-answering product that can that can understand the information in these articles and answer some simple questions related to those articles.

Proposed Solution

We plan to use Natural Language Processing 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 POS tags, lemmas, synonyms, hypernyms, meronyms, etc. for every sentence, and use Solr tool to store & index all this information. We'll extract the same features from the question and form a Solr search query. This query will fetch the answer from the indexed Solr objects.

Implementation Details

Programming tools:

- Python (*version*: 3.7.3) - Spyder
- Apache Solr (*version*: 8.0)
- NLTK library (*version*: 3.2.5)
- Spacy library (*version*: 2.0.13)

Architecture:

We shall cover our architecture (shown in figure 1) in the following 7 steps:

Step 1:

- Read all the sentences from the given corpus.
- Extract the following NLP features from each sentence – word tokens, lemmas, stems, synonyms, hypernyms, hyponyms, holonyms, meronyms, named entities, dependency parsed tree.
- We used Spacey library to create a dependency parsed tree of the sentence & stored it in a list.

Step 2:

- Send the sentence & it's extracted features to Apache Solr for indexing.
- Each indexed object is a list of key value pairs where each key is an NLP feature (ex. Synonyms, hypernyms, etc.) & its value is the stored in csv format.

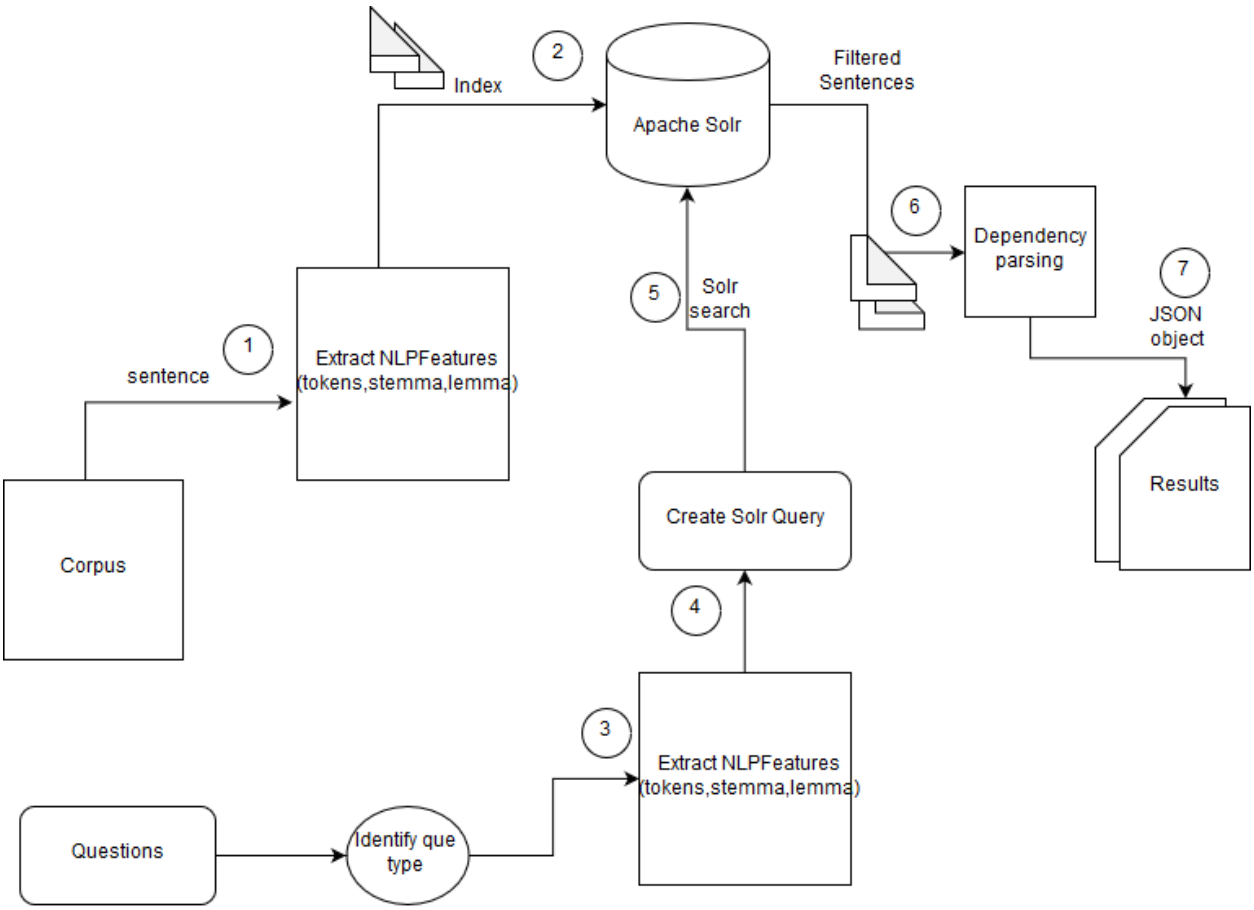

Figure 1: Architecture Diagram

- Solr has an internal synonyms.txt file which accepts csv values of words that are not common & specific to our domain.
- Solr considers these values as synonyms when indexing and querying.
  Ex.: UTD, The University of Texas at Dallas, UT Dallas.
- This can be achieved by making a configuration change in managed-schema file in Solr's directory as shown in figure 2:

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100" multiValued="true">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" words="stopwords.txt" ignoreCase="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" words="stopwords.txt" ignoreCase="true"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Figure 2: managed-schema.xml

- At the end of this step, the entire corpus would be indexed and stored in Solr, ready for querying.
- Figure 3 shows a screenshot of some features on an indexed sentence in Solr:


```
"name":["LaurenePowellJobs.txt"],
"sentence":["Funding for XQ comes from Powell Jobs' Emerson Collective."],
"word_tokens":["Funding",
  "XQ",
  "comes",
  "Powell",
  "Jobs",
  "Emerson",
  "Collective"],
"POS_tags":["('Funding', 'VBG')",
  "('XQ', 'NNP')",
  "('comes', 'VBZ')",
  "('Powell', 'NNP')",
  "('Jobs', 'NNP')",
  "('Emerson', 'NNP')",
  "('Collective', 'NNP')"],
"lemmatize_word":["Funding",
  "XQ",
  "come",
  "Powell",
  "Jobs",
  "Emerson",
  "Collective"],
"rootOfSentence":["comes"],
"synonymns_list":["{'fare', 'arrive', 'get', 'occur', 'number', 'hail', 'funding', 'financial_sup
"hypernyms_list":["{'hoard', 'develop', 'ensue', 'locomote', 'commit', 'arrive', 'invest', 'occur
"hyponyms_list":["{'vocation', 'salt_mine', 'metier', 'settle', 'average_out', 'draw_in', 'berth'
"meronyms_list":["{'sperm', 'spermatozoan', 'spermatozoon', 'sperm_cell'}"],
"holonyms_list":["{'Old_Testament', 'Ketubim', 'Hagiographa', 'Writings'}"],
"entities_list":["XQ",
```


Figure 3: Solr Object

Step 3:

- The program requires questions to be saved in a .txt file & it's path should be passed as a parameter while running the program.
- The questions can be of 3 types: Who, When & Where.
- The type of question is used determine the named-entity type of answer required (NER: Named Entity Recognition).

Question type	Required NER type of Answer
Who	PERSON, ORG
When	TIME, DATE
Where	LOC, GPE

- NLP features are extracted for each question like in step 1.

Step 4:

- Solr accepts query in key-value format and it also supports logical operators like AND, OR.
- We create a concatenated query of the extracted NLP features from the question.
- The motivation is to create a query which will have a greater match score with the required sentence in Solr.
- By using NLP features we increase the chances of matching in cases where the exact word in the question doesn't occur in the sentence stored in Solr.
  Ex: If the question has token: 'founded' but it's answer sentence has token: 'established', the query would still be able to match them as they would be present in the synonyms list.
- Some features are more probable to give better matches and they are given preference over others by adding boosting weights to them.
- We tried various combinations of boosting and found required NERs, word tokens & NERs gave best results when boosted.
- A sample query is shown in figure 4:

```
entity_labels_list:("PERSON","ORG" )^10 AND
((word_tokens:Who,founded,Apple,Inc)^10 AND
(lemmatize_word:Who,founded,Apple,Inc) AND
(synonymns_list:apple,orchard_apple_tree,set_up,ground,plant,Malus_pumila,
(hypernyms_list:United_Nations_agency,initiate,UN_agency,open,pioneer,fals
(hyponyms_list:build,eating_apple,crabapple,crab_apple,name,nominate,const
(meronyms_list:apple) AND
(holonyms_list:apple,orchard_apple_tree,Malus_pumila) OR
(entities_list:Apple Inc.)^20 O
(stemmatize_word:who,found,appl,inc))
```

Figure 4: Solr sample query

Step 5:

- A connection is opened to Solr and query is passed which returns a list of Solr objects.
- These Solr objects contain the best possible matches that Solr found for the given query.
- They are arranged in the descending order of the match score which Solr handles internally.
- Every object contains the same features that were indexed in Step 2. This helps us to extract any information about these sentences without processing them anymore, thus saving computational time.

Step 6:

- Top 5 results for every search query are taken to extract the answer.
- Best possible sentence is selected from them using Dependency Parsed tree present in Solr object of the sentence.
- The required answer is extracted from the sentence using the dependency parsed tree tags & NERs. Ex: for WHEN questions, tokens with DATE or TIME tags are chosen.

Step7:

- The extracted results from Step 6 are stored in a JSON format as follows:

```
{
"Question": "question string",
"answers":{
//answers to question here
},
"sentences":{
//supporting sentences containing answers to question here
},
"documents":{
//supporting Wikipedia documents containing answers to
question here
}
}
```

- One JSON object is created for every question. They are saved in a JSON array & dumped into 'answers.json' file.
- Figure 5 shows a screenshot of answers.json

Results & error analysis:

Figure 5 shows a screenshot of our results:

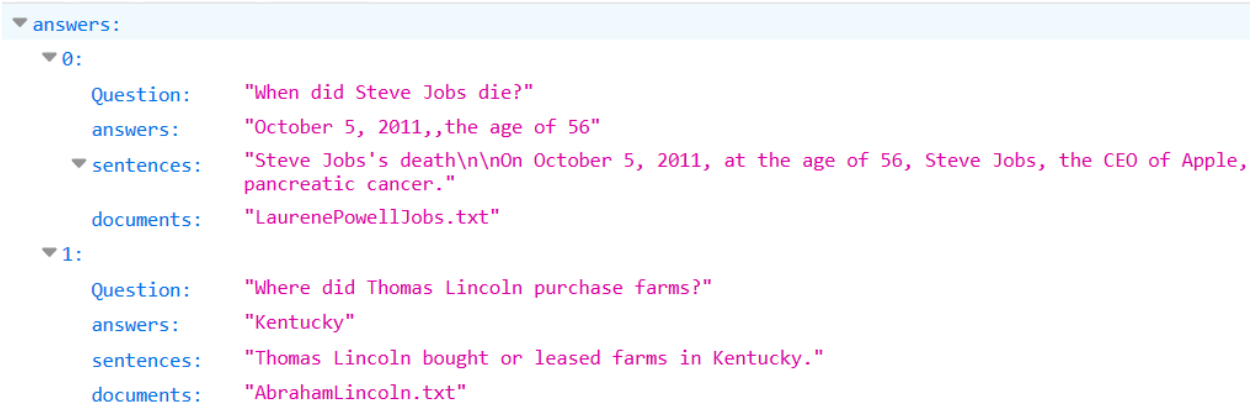

Figure 5: Sample results in answer.json

Problems encountered:

- Indexing took long time when we tried to hit Solr for each and every sentence. We resolved it by indexing the entire document (list of sentences) at once.
- The words utd, The Univ of Texas at Dallas, The UT Dallas are used interchangeably in the questions or corpus. We resolved it by making these words synonyms in the synonyms.txt file.
- Getting the required sentences in top 5 results in Solr was challenging. We resolved it by using the boosted weights for few features (entities, word_tokens and required_entities)

Pending Issues:

- Getting the required sentence as 1st result of the Solr search query.
- Accuracy can be improved in selecting the correct sentence from the list of Solr results.
- Extracting the correct phrase (answer) from the result sentence if it has more than one NERs.

Potential Improvements:

- Pronoun resolution could be a potential improvement to extract answer from the sentences which has pronouns in the places of subject/object.
- Refining the Solr query such that it selects the correct sentence from the list of Solr results.
