Frequently Asked Questions (FAQs) semantic matching

application

CS 6320 Natural Language Processing – Spring 2018

**Group:** Parse me if you can

*Divya Kishore (dxk164430@utdallas.edu)*

*Pranav Bhatia (pxb163030@utdallas.edu)*

# Introduction

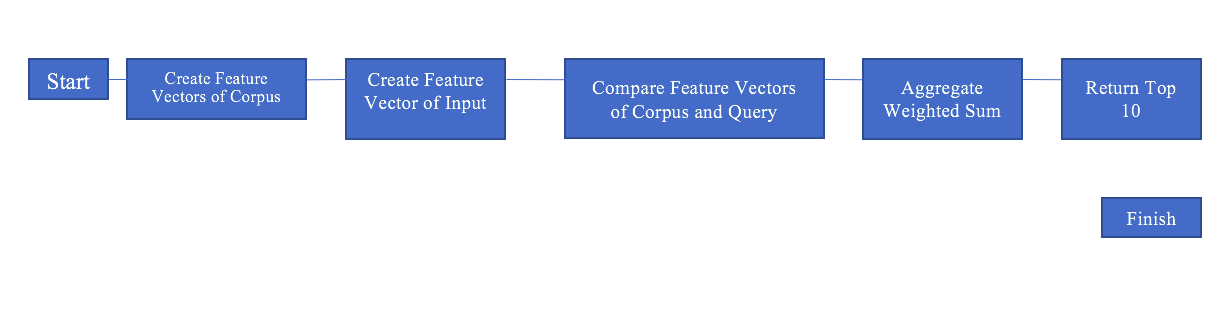
The aim of the project was to collect 50 Frequently Asked Questions(FAQs) on any topic and implement a semantic matching application using standard NLP Techniques. The FAQ’s that we collected was based on Regulations Relating to OPT Processing.

# Programming Tools Used

For our application we have used the following tools –

1. **Python:** The application was built using Python programming language.
2. **Flask:** Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. We used Flask to host our application and create a basic UI for entering queries. We used front-end technologies like HTML, CSS, jQuery and JavaScript for creating the UI.
3. **Stanford Core NLP:** Stanford CoreNLP provides a set of human language technology tools. It can give the base forms of words, their parts of speech, whether they are names of companies, people, etc.,
4. **Wordnet:** WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.

# Architectural Diagram



# Task-1: Create a corpus of 50 FAQs and Answers

For this task, we created a corpus of 50 most popular FAQs and Answers of “Regulations Relating to OPT Processing”.

This was available at <https://www.uscis.gov/sites/default/files/files/article/OPT_4Apr08.pdf>.

Cleaning the corpus:

1. Duplicate questions were discarded.
2. Questions with answers longer than 5 full sentences were discarded as the extra sentences added negligible value to the training corpus with a training and testing time overhead.
3. Removed tables, bullets, special characters if found as they didn’t provide any useful information, and also for our parser to get the information from the corpus easily.
4. Removed hyperlinks, if any.

# Task-2: Finding the best FAQ question from corpus – given an input question

We implemented a simple bag of words of the input question and matched it with bag of words of each of the FAQs in our corpus. The corpus questions were then ranked as per the two count values - the first count value counted only the unique occurrences of a particular word, while the second count was kept to track of how many times the words had occurred in the answer. We sorted the answers in a descending order - keeping primary key of sorting as first count(unique occurrences) and secondary key as second count(repeated occurrences).

Running it on the following set of questions, we got the following results –

1. Am I eligible for OPT? [Direct Question – similar to what we have in corpus word by word. Original Question: Q3. Am I eligible for OPT?]
2. When can I apply for OPT? [Direct Question. Original Question: Q5. When can I apply for OPT?]
3. Should I begin work for no salary? [Changing sense. Q43. Can I start working without pay while waiting for my EAD?]
4. Apply when OPT for can? [Jumbled words. Original Question: Q5. When can I apply for OPT?]
5. Why was my request for OPT cancelled from SEVIS? [Taken some part of answer. Original Question: Q14: May I cancel my OPT application once I have applied?]
6. Is E-Verify legal for employers? [Taken some part from answer. Original Question: Q11: How can I tell if my employer is signed up for E-Verify?]
7. Does my wife and kid get OPT too? [Changing spouse->wife and children->kid. Original Question: Q25. Does my spouse or children get a new OPT I-20 too?]
8. I misplaced my EAD what documents I require to make another one? [Changing sense of some words. Original Question: Q30. I lost my EAD. How can I get it replaced?]
9. I have completed all my forms, when can I start to work? [Lemmatized Sentence. Q36. Do I need to complete any forms with my employer to begin working?]
10. Could I goes to school on OPT? [Lemmatized Sentence. Original Question: Q41. Can I go to school while on OPT?]

So we get the following results – (Screenshots are attached in Appendix I)

|  |  |  |
| --- | --- | --- |
| Input question | Expected Answer | Expected Answer Rank and MRR |
| 1 | Q3 | Second MRR =0.5 |
| 2 | Q5 | First MRR =1 |
| 3 | Q43 | Eighth MRR =0.125 |
| 4 | Q5 | First MRR =1 |
| 5 | Q14 | First MRR =1 |
| 6 | Q11 | First MRR =1 |
| 7 | Q25 | First MRR =1 |
| 8 | Q30 | First MRR =1 |
| 9 | Q36 | Not in top 10 MRR =0 |
| 10 | Q41 | First MRR =1 |

MRR Value for the set of questions is = 7.625/10

# Task-3: Finding features for our corpus questions

In the problem statement, we were given with the 7 main features. We have broken those 7 into 12 features that we will extract from the corpus questions and later train them on. Those features are –

1. **sentence tokenizer:** Breaking down the corpus text into multiple sentences.
2. **words tokenizer:** Breaking down the corpus into words.
3. **words without stop words:** Removing all the stop words from out text. Examples of stop words would be the, a, an, but, and, etc.
4. **words with stemming:** Changing the original words in the corpus to their stemmed form. Eg - stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". For our code, we used Snowball Stemmer.
5. **words with lemmatization:** Changing the original words in the corpus to their lemmatized form. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk' is the lemma for all of the above words.
6. **words with pos tags:** Getting the part of speech tags of the words in corpus. Eg -
7. **parse tree string:** Using dependency parsing of the sentences to get the important information about it.
8. **hypernyms list:** Generating hypernyms of each word in corpus.
9. **hyponyms list:** Generating hyponyms of each word in corpus.
10. **meronyms list:** Generating meronyms of each word in corpus.
11. **holonyms list:** Generating holonyms of each word in corpus.
12. **synonyms list:** Generating synonyms of each word in corpus.

(Screenshots are attached in Appendix II)

# Task-4: Finding the best FAQ question from corpus – given an input question using the NLP features extracted in Task 3.

We studied the results of task 2 to conclude that the features extracted in task 3 could be weighted to give better results, that would give significantly better results than a simple BOW technique. Weights of feature vectors were tuned after multiple iterations of trial. We assigned the following weights to these features – the range of the weights we assumed is from [1-10]

1. **sentence tokenizer: 10**

Assigning sentence tokenizer higher priority because if we get an exact match with one of the sentences in the corpus, that question should be the one we are looking for.

1. **words tokenizer: 10**

Same reason as above. If we get a good amount of input words in that particular corpus, greater chance of that FAQ being correct.

1. **words without stop words: 10**

Stop words add little value to the corpus, matching words without stop words gave very accurate results.

1. **words with stemming: 6**

Stemming words often give results that are not real words in the dictionary, hence lower weight.

1. **words with lemmatization: 10**

The base form of words match identically, giving accurate results.

1. **words with pos tags: 6**

POS tags were a useful feature as it tells us the context in which a particular word was used, hence a moderately high weight.

1. **parse tree string: 3**

Considering that the very same sentence can give multiple parse trees, we found this feature to not be of much importance.

1. **hypernyms list: 3**

Hypernyms did not seem like a relevant feature for our corpus, hence the low weight.

1. **hyponyms list: 3**

Like hypernyms, hyponyms were a feature set that were better to be ignored.

1. **meronyms list: 3**

Meronyms is a feature vector that could be useful for a corpus that did not have as many technical terms as the corpus used by us.

1. **holonyms list: 3**

Holonyms, like meronyms were not useful for a corpus that had very specific terminology.

1. **synonyms list: 7**Use of synonyms captures the essence of the language.

Running it on the following set of questions, we got the following results –

1. Am I eligible for OPT? [Direct Question – similar to what we have in corpus word by word. Original Question: Q3. Am I eligible for OPT?]
2. When can I apply for OPT? [Direct Question. Original Question: Q5. When can I apply for OPT?]
3. Should I begin work for no salary? [Changing sense. Q43. Can I start working without pay while waiting for my EAD?]
4. Apply when OPT for can? [Jumbled words. Original Question: Q5. When can I apply for OPT?]
5. Why was my request for OPT cancelled from SEVIS? [Taken some part of answer. Original Question: Q14: May I cancel my OPT application once I have applied?]
6. Is E-Verify legal for employers? [Taken some part from answer. Original Question: Q11: How can I tell if my employer is signed up for E-Verify?]
7. Does my wife and kid get OPT too? [Changing spouse->wife and children->kid. Original Question: Q25. Does my spouse or children get a new OPT I-20 too?]
8. I misplaced my EAD what documents I require to make another one? [Changing sense of some words. Original Question: Q30. I lost my EAD. How can I get it replaced?]
9. I have completed all my forms, when can I start to work? [Lemmatized Sentence. Q36. Do I need to complete any forms with my employer to begin working?]
10. Could I goes to school on OPT? [Lemmatized Sentence. Original Question: Q41. Can I go to school while on OPT?]

So we get the following results – (Screenshots are attached in Appendix III)

|  |  |  |
| --- | --- | --- |
| Input question | Expected Answer | Expected Answer Rank and MRR |
| 1 | Q3 | First MRR =1 |
| 2 | Q5 | First MRR =1 |
| 3 | Q43 | First MRR =1 |
| 4 | Q5 | First MRR =1 |
| 5 | Q14 | First MRR =1 |
| 6 | Q11 | First MRR =1 |
| 7 | Q25 | Second MRR = 0.5 |
| 8 | Q30 | Fifth MRR =0.2 |
| 9 | Q36 | First MRR =1 |
| 10 | Q41 | First MRR =1 |

MRR Value for the set of questions is = 8.7/10

# Problems Faced –

1. Choosing the correct stemmer for our problem –

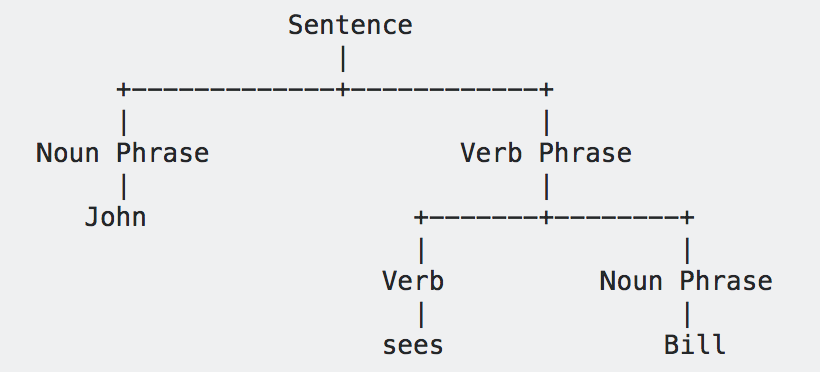
The three major stemming algorithms in use today are Porter, Snowball and Lancaster.

* Porter: Most commonly used stemmer without a doubt, also one of the most gentle stemmers.
* Snowball: It is universally regarded as an improvement over porter. It has faster computation time than porter and also has a fairly large community around it.
* Lancaster: Very aggressive stemming algorithm. The fastest algorithm all three.

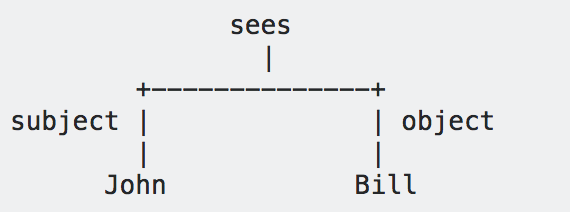
In our case, we went with Snowball as it was fairly less aggressive and the stemmed representations are usually fairly intuitive to a reader. With Lancaster many shorter words became totally obfuscated. Also since our corpus wasn’t that huge, we were not worried about the speed.

1. Using which type of parse tree – constituency vs dependency1

* A constituency parse tree breaks a text into sub-phrases. Non-terminals in the tree are types of phrases, the terminals are the words in the sentence, and the edges are unlabeled. For a simple sentence "John sees Bill", a constituency parse would be:



* A dependency parse connects words according to their relationships. Each vertex in the tree represents a word, child nodes are words that are dependent on the parent, and edges are labelled by the relationship. A dependency parse of "John sees Bill", would be:



In our case, we choose Dependency parsing as a feature since it gives us more information about a sentence, than the constituency one. We extract important concepts from the sentence, such as the subject, object, ROOT, compound, punct, etc.

# Pending Issues -

There are currently no pending issues.

# Potential Improvements –

1. Extracting more features of a document for example – using word2vec can significantly improve our results.
2. To keep it simple, we used weights as whole numbers between 1-10. Maybe using real numbers, could have improved our results.
3. Completely eliminating some features such as meronyms, parse trees rather than giving them a small weight.

# References –

1. <https://stackoverflow.com/questions/10401076/difference-between-constituency-parser-and-dependency-parser>