Semantic Search Application

CS 6320-501 Natural Language Processing – Fall 2017

**Group:** GASBDV (General Anakin Skywalker Becomes Darth Vader)

*Stephen Blystone (smb032100@utdallas.edu)*

*Amit Gupta (axg162930@utdallas.edu)*

*Deepan Verma (dxv160430@utdallas.edu)*

Table of Contents

[Introduction 3](#_Toc500326440)

[Programming Tools Used 3](#_Toc500326441)

[Requirements 3](#_Toc500326442)

[Task 1: Corpus of News Articles 4](#_Toc500326443)

[Description 4](#_Toc500326444)

[Selection of Corpus 4](#_Toc500326445)

[Cleaning the Corpus 4](#_Toc500326446)

[Querying the Corpus 5](#_Toc500326447)

[Task 2: Shallow NLP Pipeline 5](#_Toc500326448)

[Description 5](#_Toc500326449)

[Proposed Solution 5](#_Toc500326450)

[Architectural Diagram 5](#_Toc500326451)

[Results and Error Analysis 6](#_Toc500326452)

[Task 3: Deeper NLP Pipeline 7](#_Toc500326453)

[Description 7](#_Toc500326454)

[Proposed Solution 7](#_Toc500326455)

[Architectural Diagram 7](#_Toc500326456)

[Results and Error Analysis 8](#_Toc500326457)

[Task 4: Improving Pipeline Results 8](#_Toc500326458)

[Description 8](#_Toc500326459)

[Proposed Solution 9](#_Toc500326460)

[Architectural Diagram 9](#_Toc500326461)

[Results and Error Analysis 10](#_Toc500326462)

[Issues encountered during the Project 12](#_Toc500326463)

[Pending Issues 12](#_Toc500326464)

[Potential Improvements 13](#_Toc500326465)

[Conclusion 13](#_Toc500326466)

# Introduction

This project is a culmination of many of the topics we have learned this semester. We were asked to design and implement a semantic search application using standard NLP tools, features, and techniques.

# Programming Tools Used

For our program we used Python as our programming language, Apache SOLR to store our data, PySOLR as our Python API into SOLR, NLTK for most of the NLP feature extraction, and Stanford Core NLP to extract the head words. We also installed the Stanford Core NLP server locally to improve response time and to not overload Stanford’s server with queries.

# Requirements

* Training data must be in the “TrainingData” folder.
* Stanford Core NLP version 3.8.0
  + Download:

Full Application: <http://nlp.stanford.edu/software/stanford-corenlp-full-2017-06-09.zip>

English jar file: <http://nlp.stanford.edu/software/stanford-english-corenlp-2017-06-09-models.jar>

* + JAR file must be in unzipped directory of Core NLP.
  + To run (from the unzipped directory):

java -mx4g -cp "\*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout 15000

***NOTE****: You may have to use -mx3g or -mx2g or -mx1g depending on your computer memory*

* Apache SOLR 7.1.0
  + Download:

Full Application: <http://apache.mirrors.lucidnetworks.net/lucene/solr/7.1.0/>

* + To create cores (from bin directory):

solr.cmd create -c part2core

solr.cmd create -c part3core

* + To run (from bin directory):

solr.cmd start

* Python 3.5.2
  + Download: <https://www.python.org/downloads/release/python-352/>
* Python (3.5.2) Modules:
  + NLTK (3.2.5)

Install: pip install nltk

Download required NLTK components:

python

import nltk

nltk.download(‘punkt’)

nltk.download(‘wordnet’)

nltk.download(‘averaged\_perceptron\_tagger’)

* + PySolr (3.6.0)

Install: pip install pysolr

* + StanfordCoreNLP (3.7.0.2)

Install: pip install stanfordcorenlp

# Task 1: Corpus of News Articles

## Description

For the first task we had to create a corpus of news articles that contained at least 1,000 articles and 100,000 words.

## Selection of Corpus

We selected the Reuters-21578 Benchmark Corpus, ApteMod version as our corpus to use for this project. This is publicly available for “research purposes only” at <https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/corpora/reuters.zip>

We chose this because it contains a large set of news articles covering a variety of topics.

## Cleaning the Corpus

The corpus contained many duplicate articles, as well as articles that were subsets of other articles. This caused problems when we were running queries because we would get duplicate results from different articles. We wrote a script to identify which articles had identical content and removed all but one copy of each of those articles. This removed the duplicate articles from our corpus. We wrote another script to identify articles with the same first few lines and removed all but the longest version of the article. One article might only contain the first paragraph, while a second article might contain the first 2 paragraphs, and the third article might contain the full content. With these scripts we were able to clean our corpus.

## Querying the Corpus

After testing and studying the corpus we came up with the following 10 queries:

1. “G-7 support for the Paris Agreement”
2. “Selling cows to Indonesia”
3. “Increase loans in Virginia”
4. “Thailand tin export company”
5. “Aramco field oil reserves increase”
6. “Australia is more competitive than the U.S.”
7. “China buying grain”
8. “Shareholders vote on buyout”
9. “Indonesia importing oil”
10. “Australia's economy performs well”

# Task 2: Shallow NLP Pipeline

## Description

Task 2 requires us to implement a shallow NLP pipeline that indexes and queries SOLR using sentences that are converted into word vectors. We must evaluate the top 10 returned sentences for 10 queries.

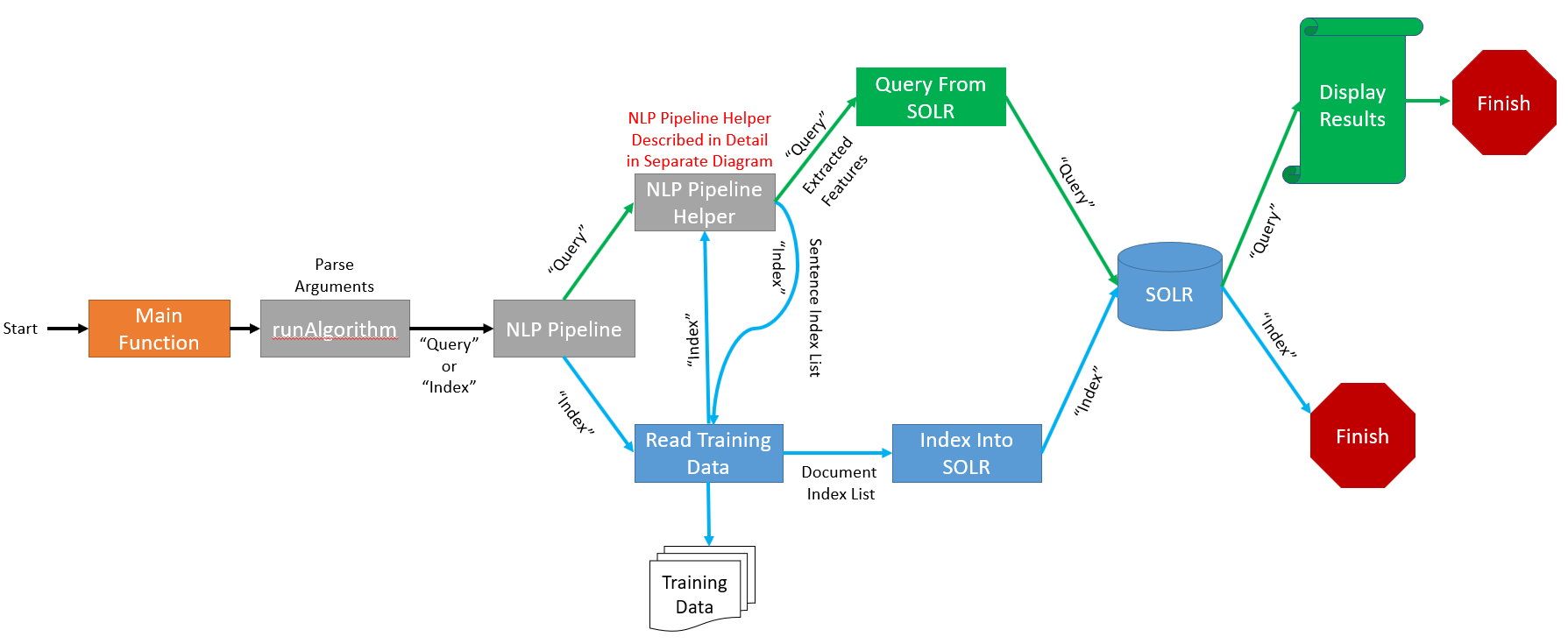
## Proposed Solution

Our proposed solution is to create an NLP Pipeline that is passed an argument indicating whether we are going to “index” data into SOLR, or “query” data from SOLR. If we “index”, then we read in the Training Data and send that information to an NLP Pipeline Helper function that segments each input into sentences and tokenizes words from those sentences. NLP Pipeline Helper will then return the Sentence Index List for each document. The overall document index list is then sent to SOLR for Indexing.

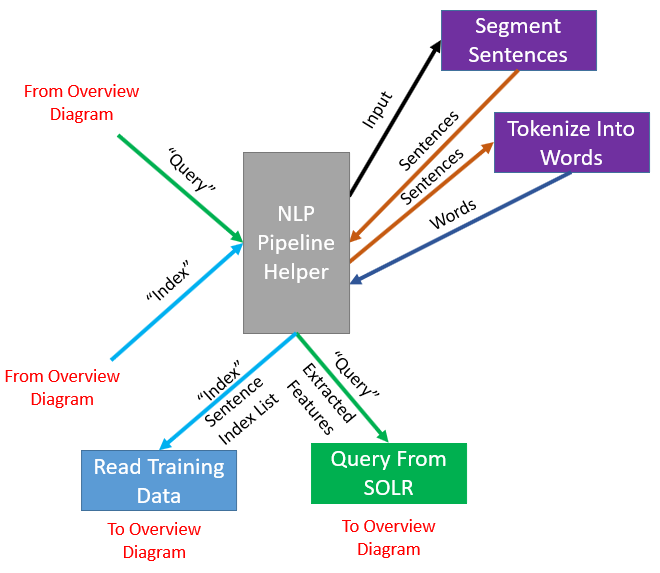
If we “query”, then we call NLP Pipeline Helper function to segment our query into sentences and tokenize words from those sentences. We then send the extracted features to SOLR for querying, and then display the top 10 results to the screen.

## Architectural Diagram

Below is the high-level overview diagram:



Here is the detailed diagram for NLP Pipeline Helper:



## Results and Error Analysis

We used Mean Reciprocal Rank (MRR) to grade our “query” results. To grade the results for Task 2, we manually looked at each sentence that was returned from SOLR and determined whether it was a valid sentence for our query. The MRR for Task 2 was 0.675

Here are the ranks for each query and overall MRR:



# Task 3: Deeper NLP Pipeline

## Description

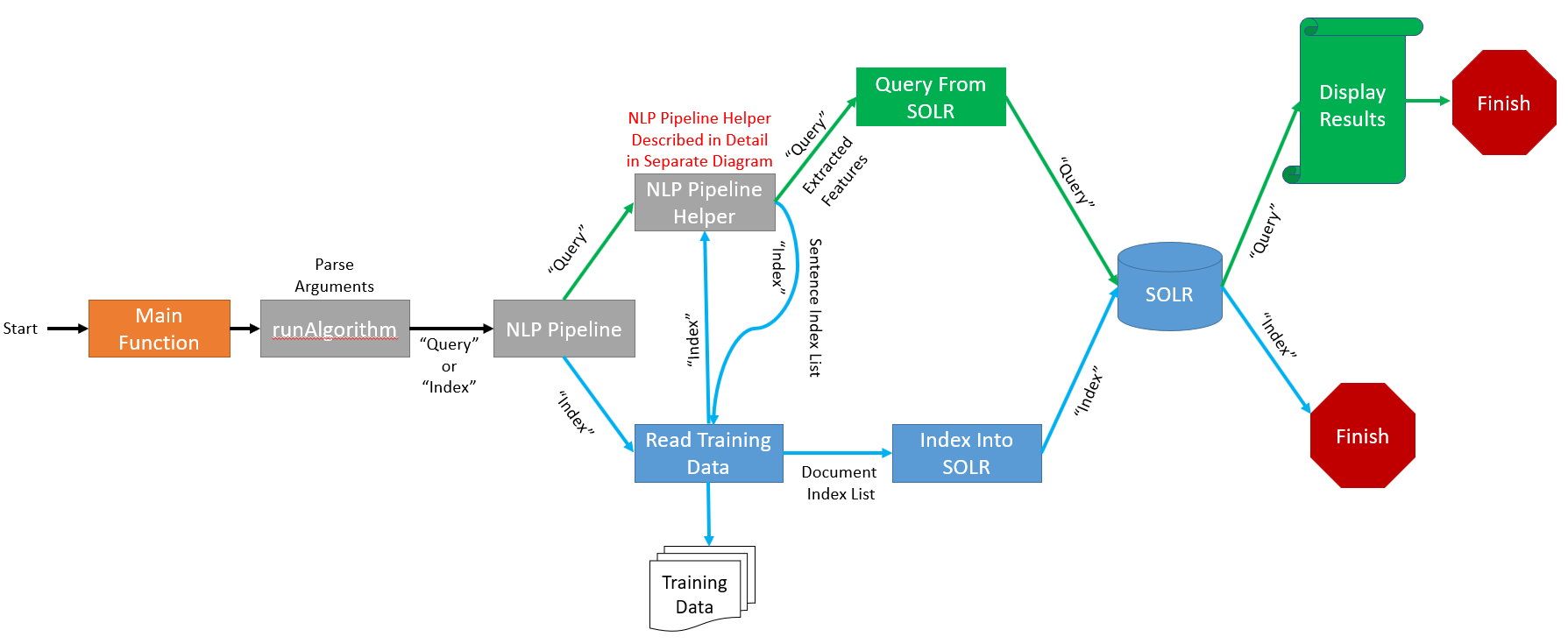
Task 3 requires us to implement a deeper NLP pipeline that indexes and queries SOLR using sentences that are converted into word vectors, lemmas, stems, POS tags, head words, hypernyms, hyponyms, meronyms, and holonyms. We must evaluate the top 10 returned sentences for 10 queries.

## Proposed Solution

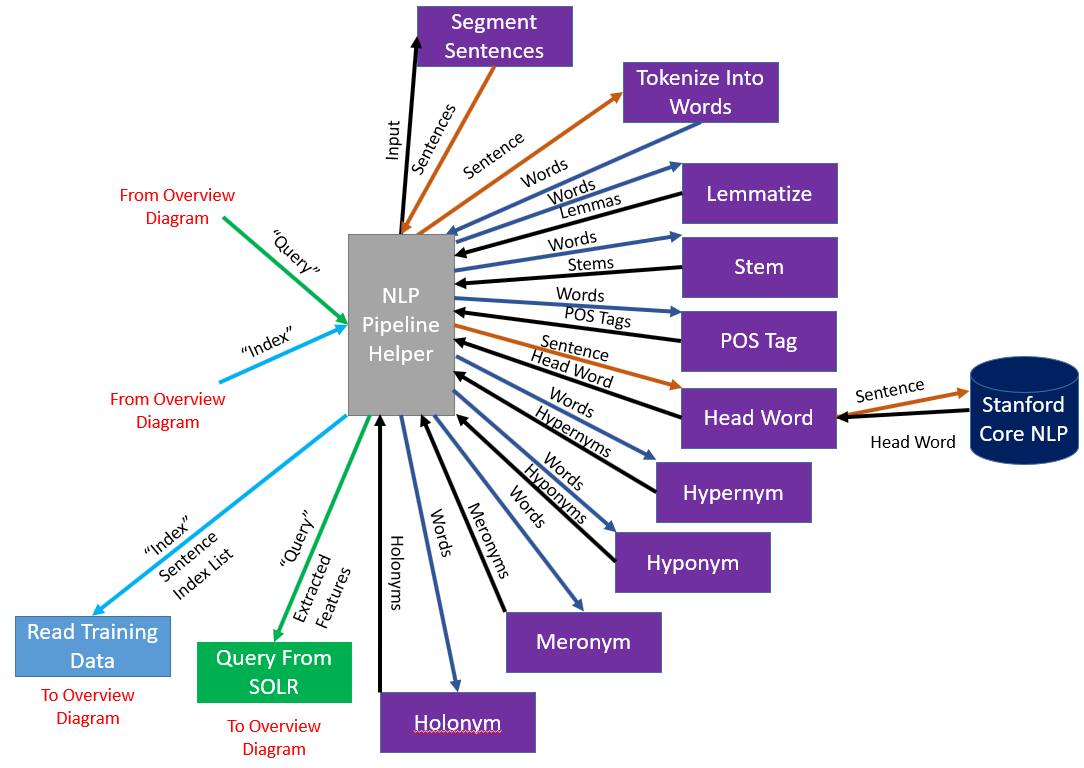
Our proposed solution remains the same as Task 2, except for NLP Pipeline Helper. NLP Pipeline helper now performs the following tasks: segment input into sentences, tokenize a sentence into words, lemmatize words into lemmas, stem words into stems, POS Tag words into POS Tags, use Stanford Core NLP server to extract the head word from a sentence, get hypernyms from words, get hyponyms from words, get meronyms from words, get holonyms from words. As in Task 2, NLP Pipeline Helper sends the extracted features to the “Query From SOLR” function if we are “query” or returns the sentence index list back to the “Read Training Data” function if we are “index”.

## Architectural Diagram

Below is the high-level overview diagram:



Here is the detailed diagram for NLP Pipeline Helper:



## Results and Error Analysis

Like Task 2, we used MRR to grade our results. For Task 3 we also manually graded the results from each query. Our MRR for Task 3 is 0.26595 which is a significant drop from our Task 2 result of 0.675.

Here are the ranks for each query and overall MRR 

# Task 4: Improving Pipeline Results

## Description

Task 4 requires us to improve our results from Task 3 by selecting which features to use and by adjusting the importance of the features (by assigning them weights).

## Proposed Solution

For Task 4 we added in global FLAG and WEIGHT variables for each indexed feature and used command-line arguments to set the FLAGs and WEIGHTs. The FLAG values are either True or False depending on if we are using the feature or not. The WEIGHTs are all initialized to 1, but can be changed using the command-line arguments.

Instead of manually typing out each command-line FLAG and WEIGHT argument, we created a wrapper program that reads in a “TestInput” file and calls projectTask3.py using the parameters for each test in that file. There is a sample of the “TestInput” file in the Architecture Diagram below. Using this wrapper program, we could add multiple tests into the same file and easily test various feature combinations and weights. The wrapper program produces 3 output files:

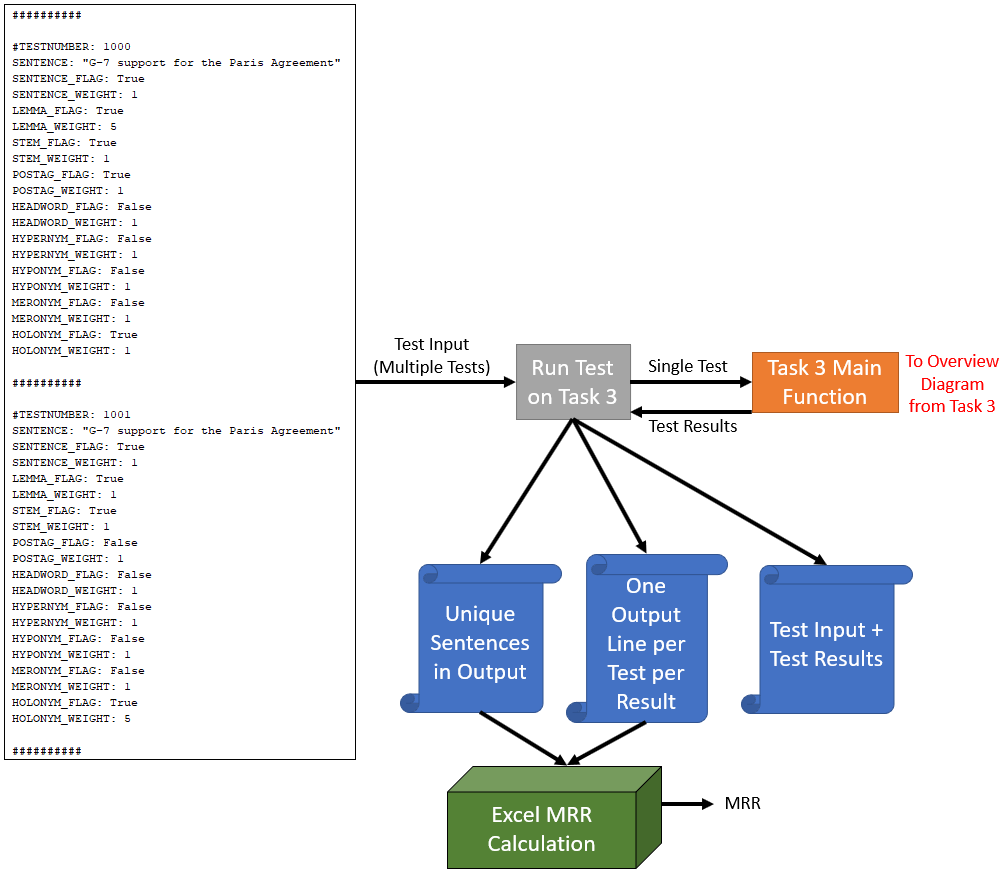
1. A file that contains all the unique sentences. This file contains data for the document-sentence ID and the sentence. This was used for us to easily grade which sentences were good for a query.
2. A file that contains one output line per test per result. This file contains data for the test number, the rank in the output, the document-sentence ID, and the sentence. If we had 2 tests in our “TestInput” file and each test produces 10 sentence results from SOLR, then there would be a total of 20 rows in the output. This file was used in calculating the MRR by identifying the rank of the first “good” result for a given test.
3. A file containing the raw test input for a given test, as well as the raw output for that test. This was used so we could see what the FLAG and WEIGHT values were for a test number.

After performing some testing, we realized that certain features worked better for certain types of target sentences and queries.

We wrote another script to create a “TestInput” file for each query that contained every combination of FLAG variables. We left all WEIGHT variables set to 1. We did this to give us a baseline for which features seemed to work well with our corpus and our queries. Using this method, we found some combinations of features that had the highest MRR. See the Results section for more details.

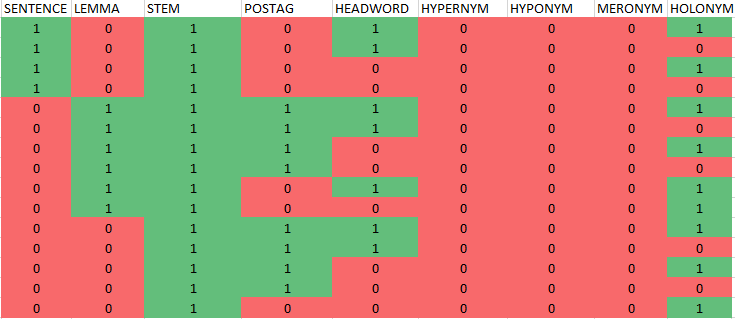
## Architectural Diagram

The Architecture Diagram is the same as Project Task 3 diagrams, except for the wrapper program which is shown below:



## Results and Error Analysis

As mentioned in our Proposed Solution, we tested all combinations of FLAGs for each feature to see what worked in our corpus. Below are the features used in each of the top 15 MRR results:



This made it easy for us to tell that hypernym, hyponym, and meronym were not helpful with the queries we were using. We also saw that stem was important. Comparing the actual scores, we found that head word did not make a difference. The same feature combinations with head word had the same MRR value as the same feature combination without head word.

Most of the results seem to use either sentence by itself or lemma and POS-tag together. However, our intuition told us that the words in the sentence were still important, and should be able to be used in conjunction with lemma and POS-tag if we adjusted the weights correctly.

Our final weights were:

* Sentence (tokenized words): 1
* Lemmas: 100
* Stems: 50
* POS-Tags: 50
* Holonyms: 50

This produced a MRR value of 0.9, which is significantly higher than both the Task 2 score of 0.675 and the Task 3 score of 0.26595. Below are the ranks and MRR comparing Task 2, 3, and 4:



The command used to reproduce these results is:

python projectTask3.py --userInput "Australia is more competitive than the U.S."   
--testing --sentenceFlag --sentenceWeight 1 --lemmaFlag --lemmaWeight 100   
--stemFlag --stemWeight 50 --posTagFlag --posTagWeight 50 --holonymFlag   
--holonymWeight 50

As mentioned above, we went against the results of our script that tested all combinations of features, and decided to include sentence (tokenized words), lemma, stem, POS-Tag and holonym into our final result.

* We decided that sentences (tokenized words) should be included because if we have an exact match between query and indexed sentence in SOLR, we want to extract that. Based on our testing we found the other features were more important, so we set sentence weight to 1.
* We gave lemmas the highest weight, 100, because some of our queries did not use the exact word that was in the indexed sentence, but they had the same lemma. For example, “selling” in our query and “sell” in the indexed sentence.
* We found stems to be useful for similar reasons to lemmas. In our queries though we didn’t have as many instances where our query and target sentence had stems but not lemmas. Because of this we set the weight lower than lemma to 50.
* POS-tags were also useful because they compared the tags on the words, so we set to 50.
* Holonyms were also useful. We tried running the same query using holonyms and without holonyms, and the query using holonyms scored higher. We found 50 to be the ideal weight.

# Issues encountered during the Project

We encountered several issues during the project. The first was mentioned earlier in Task 1 about the duplicate articles and was resolved using 2 scripts, also mentioned in Task 1. Another issue we encountered was the Stanford Core NLP server sometimes ran out of available ports to accept our query and respond back. It took a while to find this issue. The solution was to check for an exception when calling the Core NLP server, sleep for 5 seconds, and then try again. After 5 seconds the server released some of its ports and they were made available again.

# Pending Issues

There are currently no pending issues.

# Potential Improvements

We could become more granular in selecting weights. In our project we just tested weights in increments of 5’s and 50’s. We could test smaller increments and see if we improve our test results.

We also found that different feature combinations and weights worked better for different kinds of sentences. To improve on this, we could write a wrapper script that determines the type of sentence and query, dynamically sets the parameters, and call our projectTask3.py script using the appropriate parameters.

# Conclusion

This project provided us an excellent opportunity to use the knowledge gained in class and apply it to a real-world NLP application. Our results helped us to realize the importance of proper feature selection and weight selection. The drop in MRR from 0.675 to 0.26595 between Task 2 and Task 3 showed us that blindly adding more features is not necessarily a good thing. We also saw that the feature selection and weights are influenced by the corpus and the queries chosen.