Mahmood Ahmed TUID: 915598955

Feature Extraction

******DISCLAIMER: ONCE THE PROJECT IS RUN ONCE AND THE “NLTK DOWNLOAD” MENU APPEARS, SELECT POPULAR AND CLOSE THE WINDOW ONCE IT HAS COMPLETED DOWNLOADING. ON FURTHER RUNS, YOU MAY COMMENT OUT THIS LINE USING “##”

When we first received this problem I began thinking about what makes two sentences a paraphrase. Ultimately, it comes down to meaning. I broke my features into 3 main categories; quantifiable, grammatical, and definitive.

Additionally I wrote a function (**called comparison)** that would measure the similarity between two sentences given the frequency of a certain prompt in each. (E.g. if both sentences had the same number of proper prepositions, they likely convey similar meaning). I divided the minimum number of occurrences by the total between the two sentences to normalize this feature between 0 and 1. If both numbers were 0, the output was 1(very similar). If only one number was 0, I returned 1 over the total. A higher discrepancy (i.e. 0 & 8) would yield less similarity than a lower one (0 & 2).

Quantifiable features could be easily counted. Grammatical features make up most of the model.

- **Word Count**
- **Number of Capital Letters** -useful to measure the number of proper nouns in a sentence
- **Number of numbers -** useful for dates and statistics

No matter how many synonyms you change in a sentence, the meaning is often revealed through grammatical context. Here I focused on different parts of speech and hardcoded lists of these varying parts. For each **item** in each of the sets listed below, the number of occurrences in each sentence was compared using the function described above. This number was averaged over the entire set and that number was stored as the feature. Unsurprisingly, these numbers often were close to 1.

- **Similarity in punctuation**
- **Similarity in number of articles**
- **Similarity in number of conjunctions**
- **Similarity in number of prepositions**
- **Similarity in number of personal pronouns**
- **Similarity in number of demonstrative pronouns**
- **Similarity in number of interrogative pronouns**
- **Similarity in number of relative pronouns**
- **Similarity in number of reflexive pronouns**

- **Similarity in number of possessive pronouns**

The final features were concerned with the literal meanings of each word. After doing my grammatical comparisons, I parsed each input sentence down, so by the time I measured these features, only the keywords were considered for overall meaning.

- **Similarity in noun suffixes**
- **Similarity in verb suffixes**
- **Similarity in adjective suffixes**
- **Similar google search results**- this feature compares the top results of each keyword query and divides by the total # of links. If both queries were essentially the same, this returned 1.
- **Frequency of synonyms**- For each word in the first sentence, I divided the number of occurrences in the second sentence by the number of occurrences in the first. This is because if the first sentence only mentions the word "apple" once and the second does ten times, then they are likely not paraphrases. Also, I ran this function twice using all the words in the second sentence–in the event that it was much longer and still had keywords that had not been considered. The final feature was the average of these two values. Furthermore, before I multiplied the frequency of synonyms by the relative size of the two sentences using **comparison** (the function described above) in order to further account for the skew in the average when one sentence was much longer.

Libraries Used

- **Beautifulsoup4**- This package was used to handle the google results page and return an array of relevant links once a query was input
- **Requests**- This package was used to make requests to the internet.
- **Nltk**- The Natural Language Toolkit was helpful for dealing with strings. The tokenize function was used to break down the input sentences, so that punctuation appeared as separate array elements and alphanumeric inputs could be more easily compared. Also, this package was responsible for providing lists of synonyms depending on the input word.
- **Numpy**- This library was used to store data from my 2-d arrays. Once the features and golden truths had been organized once, I saved them to .npy files in order to cut down on the time complexity of the project.
- **Sklearn**- This was the main package used to create and finetune the Machine Learning model. It accepted arrays for both my features and expected results, as well as providing me with an accuracy score so that I could further improve the model given the data in the dev set.

Takeaways

- In the end, I could not apply my most promising feature (the google search comparison. This was because in order to loop through the full training sample, I needed about five hours of uninterrupted network connection. If I could redo the project, I was divide the training set down and concatenate a series of feature sets for my model training

- Perhaps it would have been better to scale all my features after creating the model and checking the accuracy. Maybe there was not enough variance because of how often I used the **comparison** function and this led my model to skew toward a positive result.
- The Logistic Regression gave me higher performance on the development set.
- I should have focused more on pre-processing. Once I had a model that was scoring 59% accuracy on the development set, I noticed that the training data was highly imbalanced. To remedy this, I used flags during my feature extraction and ignored the first 2000 samples with a positive golden truth value. I thought this would improve my model given a balanced test set.