Stephen Hullender
CIS 4526 001
October 27, 2022

# Midterm Report: Paraphrase Identification

## Data Preprocessing

In this project, there are 3 datasets that we use for training and testing, more specifically training with ‘train_with_label.txt’, tuning/validation with ‘dev_with_label.txt’, and testing with ‘test_without_label.txt’. The datasets that were given consisted of two columns of sentences, and a label that indicates whether the sentences have the same context (this attribute is excluded in the test data).

In the first step of pre-processing the data, we iterated through each line in the text files to be converted as rows in each designated Pandas DataFrame. Any rows that were not converted due to inconsistencies with delimitation or encoding issues were skipped. All rows thereafter that contained null values were also dropped, and on a rare occasion, some rows were manually eliminated for other errors while reading the DataFrames. Later in the project, it was discovered that there were inconsistencies with comparing the length of sentences for each pair (‘sentence_1’ and ‘sentence_2’).

One example of tuning that was performed to optimize the model was when trying to get the sparse matrix similarity method to work. One issue was that the time it took to run each document was slow because every individual document on one column needed to be compared to the entire corresponding column, all 3876 documents each time. In an attempt to counteract the time it took, what I discovered was that only comparing it to its corresponding document does not accurately calculate the similarity score and sometimes gave zeros. To solve this, I tried to only fetch 500 documents that would traverse down the list of documents as each individual document’s score was being calculated. This worked in reducing the time it took to get the similarity scores, but as each document was being calculated, the scores became less accurate. As a final change, I decided to start with 500 documents, but each iteration will increase the size of the window of documents being compared to. It is not guaranteed if this method actually does anything or can be explained, but it was the best way that I could get as high of accuracy scores across all documents as possible.

## Data Cleaning (Text)

For each table, being the training, validation, and testing data, all documents were converted into an array of words. The process to filter out and refine these words is to map the column to the function that was created to clean up the text. Inside this function, significant

steps are taken to ensure the words are distinct, without stop-words that are redundant in the document's value, and processed accordingly by its base meaning (i.e. lemmatizing the words to not confuse the model of distinguishing words based on plurality).

All sentences were first converted to lowercase words, then split into an array. From there, all words that were contractions, words like "can't", "he'll", or "we've", were converted to two separate words (i.e. "can not", "he will", and "we have"). Through regex methods, we wanted to get rid of any punctuation as well as other unusual characters/strings like breaks (as indicated by the HTML tag **<br />**), ampersands, or URL links. Finally, to increase the value of all words, words like "a" and "the" were taken out completely as they carried no value to the document. At best, these so-called stop-words would do nothing to the document, but it is possible that the words will count against other words that may carry more meaning when comparing the two texts.

Lemmatizing words were carried out separately through the **WordNetLemmatizer** method. As previously done, all words were to be returned as a list of words, with each column of sentences being mapped to each word to determine whether the word would be lemmatized. In practice, all words that were changed happened to change because of plurality (i.e. "changes" turned to "change"). This was different as I had expected the words to follow a root-stem model.

**Feature Selection and Design**

These algorithms and methods were used for feature selection:

- TF-IDF / bag of words model
- Cosine Similarity
- Sparse Matrix Similarity (text similarities)
- & differences and average of length between two pairs of sentences

For TF-IDF, or *Term Frequency - Inverse Document Frequency*, I used **TfidfVectorizer** and made each model for their respective column. This amounted to 6 total vectorizers, 2 of each used for the training, validation, and testing datasets, as there are 2 columns that need to be encoded and measured via TF-IDF.

A lot of issues involved making sure all the words were taken into account when comparing each sentence in each dataset. Because some words were only available in one set, when comparing text data between the two columns 'sentence_1' and 'sentence_2', we took all the features from our vectorizer and added the missing words manually. This would ensure that the shape of the matrices are not rugged or dissimilar when passing the data through the cosine similarity function. From this, we had used set methods to differentiate which words did *not* appear on each column, and added the missing words to another column's feature names. One more thing I did was for all the missing words from a column, I added however many rows were

needed for each dataset in order to assert that the number of words and documents equaled the shape of the arrays.

**Use of Models**

For each table (train, dev, test), we had to check for the similarity scores based on the following pipeline: Bag to words → TF-IDF model → *sparse matrix similarity*

The most important step is the sparse matrix similarity. This helps with comparing the corpus, or a collection of written texts, of the two columns of documents. For every document that was to be used to calculate its similarity score, the document compared with the entirety of its opposing column. More information about this can be read in section *Data Preprocessing*, third paragraph. Each time a document is read, a bag-of-words model converts the words into numbers and is read into a TF-IDF model, similar to the TfidfVectorizer we used. The sparse matrix similarity differentiates each document's value with respect to its relevance with other documents that it is being compared to.

To ensure all data would be printed and read into the Logistic Regression model correctly, all data, which consisted of our cosine similarity scores, sparse matrix similarity scores, text length average (which comes later), and text length difference (also comes later). Pickle files were also used to save the format and data type of all results when rendering both the text similarities and the comparison data. The main purpose is to reduce time as the process takes an average of 25 minutes to finish everything.

**Algorithms and Libraries**

The only machine learning algorithm that was successfully built was the Logistic Regression model. Logistic Regression seemed to be a great fit for the data provided as it is able to take multiple classes and is useful for determining true or false values, or 1 or 0 in our case. However, one disadvantage that could have been fixed with Support Vector Machine is that SVM works better with unstructured data and is not as linear, which may help with flexibility when determining whether a label should be 1 or 0.

All the imports that were necessary to conduct the project involved the following:

- sklearn (scikit-learn)
    - model_selection → **train_test_split**
    - metrics →
        - **accuracy_score**
        - **precision_score**
        - **recall_score**
        - **f1_score**
    - svm → **LinearSVC**

        - linear_model → **LogisticRegression**
        - feature_extraction
            - text → **TfidfVectorizer**
- scipy
    - **linalg**
    - spatial → **spatial**
    - metrics_pairwise → **cosine_similarity**
    - sparse → **csr_matrix**
- nltk
    - **WordPunctTokenizer**
    - corpus → **stopwords**
    - tokenize → **word_tokenize**
    - stem → **WordNetLemmatizer**
- gensim
    - **corpora**
    - **models**
    - **similarities**
- pandas
- NumPy

Other imports were used for recurring or one-time functions like timestamps, regular expressions for filtering punctuation out of text, saving data on Pickle files (for purposes of reducing time to create new data each time worksheet runs), and for checking on the Google Drive filesystem for existing files.

**Issues**

Some algorithms that were tested were not carried out due to issues with RAM. One method that seemed promising was the **Wasserstein distance**, otherwise known as *Earth mover's distance*. The reason for selecting this option was because there were better measurements in weighing the magnitude of the vectors. Cosine similarity is also not as powerful when trying to compare words that are dissimilar, such as the case with several documents that, although may have the same meaning, used completely different vocabulary from each other. Upon testing the method, from scipy, my RAM immediately flew up and caused the worksheet to crash. It was this decision that led me to purchase CoLab Pro, but after testing some time more, I eventually decided to find another solution, leading to the sparse matrix similarity method.

**Results**

Our first short test involved using Logistic Regression to split the training data by approximately 70% training and 30% testing. The results showed a decent amount of accuracy with the following metrics:

**Accuracy:** 76.2%, **precision:** 71.9%, **recall:** 76.2%, and **F1-score:** 71.8%.

The classification matrix shows the following scores after running logistic regression between the training and validation data:

| | | | |
|---|---|---|---|
| **Accuracy** | 57.5% | **Precision** | 70.3% |
| **Recall** | 57.5% | **F1-score** | 49.2% |

As mentioned, some of the rows contained uneven text caused by bad delimitation. Since this data is not similar in meaning, as the meaning of the sentences are unintelligible, I also tested the accuracy scores based on changing the gold label values for whichever pair of text were nonsensical or uneven to 0.

While correcting this, I also decided to run the classification scores through three average options: ‘weighted’, ‘micro’, and ‘macro’. Unlike the first run, which only used ‘weighted’ for the average parameter, I decided to run additional parameters to see different results. The reason for the different average values are due to the differences in which scores will be calculated on. For example, ‘weighted’ means that the score

While testing each other via train-test split, the following results appear:

- *Weighted:*
  - **Accuracy:** 76.5%, **precision:** 73.1%, **recall**: 76.5%, & **F1-score:** 72.8%
- *Micro:*
  - **Accuracy**: 76.5%, **precision**: 76.5%, **recall**: 76.5%, **F1-score**: 76.5%
- *Macro:*
  - **Accuracy**: 76.5%, **precision**: 66.8%, **recall**: 58.5%, **F1-score**: 59.2%

The following were shown for the *weighted* average values:

| | | | |
|---|---|---|---|
| **Accuracy** | 58.8% | **Precision** | 71.9% |
| **Recall** | 58.8% | **F1-score** | 51.6% |

For *micro* values:

| | | | |
|---|---|---|---|
| **Accuracy** | 58.8% | **Precision** | 58.8% |
| **Recall** | 58.8% | **F1-score** | 58.8% |

For *macro* values:

| | | | |
|---|---|---|---|
| **Accuracy** | 58.8% | **Precision** | 71.9% |
| **Recall** | 58.8% | **F1-score** | 51.6% |

In comparing the results between the training and validation data, the accuracy scores and recall scores show the same percentages. One theory is that, since recall is measured as the ratio of correct samples the model was able to recognize over all correct samples in the pool, that all the correct samples were much more distinct in being recognized compared to results that indicated no paraphrase identification (label of 0).

One more step that was taken after this was adding new features to the dataset: the average of the two document's lengths, and the difference between the length of said document pairs. The reason for this change was because after manually tweaking the data in response to incorrect delimitation, the realization that a correlation between the lengths of two documents can be detected when looking for paraphrasing. The average is to indicate the overall length of the document pair, while the difference indicates how significant the results might be for predicting the correct label.

With this implemented, the results were redone. Upon testing each other via train-test split, the following results were produced:

When testing between the training and validation set, these results are made:

- *Weighted:*
  - **Accuracy:** 76.3%, **precision:** 73.0%, **recall:** 76.3%, **F1-score:** 73.1%
- *Micro*:
  - **Accuracy:** 76.3%, **precision:** 76.3%, **recall:** 76.3%, **F1-score:** 76.3%
- *Macro*:
  - **Accuracy:** 76.3%, **precision:** 66.2%, **recall:** 59.2%, **F1-score:** 60.1%

For *weighted* values:

| | | | |
|---|---|---|---|
| **Accuracy** | 59.4% | **Precision** | 71.4% |
| **Recall** | 59.4% | **F1-score** | 52.8% |

For *micro* values:

| | | | |
|---|---|---|---|
| **Accuracy** | 59.4% | **Precision** | 59.4% |
| **Recall** | 59.4% | **F1-score** | 59.4% |

For *macro* values:

| | | | |
|---|---|---|---|
| **Accuracy** | 59.4% | **Precision** | 71.4% |
| **Recall** | 59.4% | **F1-score** | 52.8% |

**Response**

Overall, this project was interesting. Despite the challenging nature of the topic, as paraphrase identification is not particularly easy to understand, I was able to understand some of the techniques involved in getting the correct steps done. Some of the significant steps in

research involved using text encoding and frequency through TF-IDF, which was useful in tabulating all the frequencies of every possible word within the selected documents. Another form of feature selection that was discovered was through cosine similarity, which aims to measure similarity between two documents. Although there were several other options mentioned in the articles I found, such as Wasserstein distance, they either contained deep learning methods or were far from public notice and were still very abstract/hard to understand.