i.) Features

All of the features designed when building the model, including features not used in the final model. Note: all of the features used are the same as the ones in the midterm project.

Lengths: The length of sentence 1 and the length of sentence 2 as two separate features.
Length Difference: The absolute value of the difference between the two sentences.
Overlap: A ratio of shared words to unique words used to get a proportion of overlap between the two sentences.
Sentence Length Difference: The length difference as proportion. Calculated by the length of sentence 1 minus the length of sentence 2 divided by the length of sentence 1.
Sentence Length Difference 2: An alternate way of finding sentence length difference. Calculated by the reciprocal of d raised to the power of the length of sentence 1 minus the length of sentence 2. d is a constant, 0.8 was used for the value of d.
Levenshtein Distance: Used to measure the difference between the two sentences. Levenshtein distance is the minimum number of single-character edits used to change sentence 1 to sentence 2. I used the levenshtein method from the pylev library, which is an implementation that uses an iterative version of the Wagner-Fischer algorithm.
Cosine Similarity: Used to measure the similarity between two non-zero vectors of an inner product space that measures the cosine between them. First, I vectorized the sentences and then used the cosine similarity formula on the two sentence vectors. I experimented with removing stopwords from the sentences, but I observed better accuracy without removing the stopwords.
Shared Words: The number of shared words between sentence 1 and sentence 2. Found using the intersection of the set of sentence 1 and the set of sentence 2.
BLEU Score: BLEU (bilingual evaluation understudy) score is a number between 0 and 1 that measure the similarity of machine-translated text to reference translations. The formula consists of brevity penalty and n-gram overlap. I used 3 different BLEU scores, for unigrams, bigrams, and trigrams. Experimenting with higher n-values, I found that it did not help the model because most of the BLEU scores were 0. I used the bleu_score method from the nltk library.
Meteor Score: Meteor score is a metric based on the harmonic mean of unigram precision and recall. I used the meteor_score method from the nltk library.
Jaccard Similarity: Jaccard similarity measures the similarity between the 2 sentences using the shared values and the unique values. Jaccard similarity was calculated by the intersection of the set of sentence 1 and the set of sentence 2 divided by the union of the set of sentence 1 and the set of sentence 2.
NIST Score: NIST evaluates the quality of the text using machine translation. This is based off of BLEU score, but it differs because NIST score gives more weight to rarer n-grams. NIST also uses a different calculation for brevity penalty. I used the nist_score method from the nltk library.

I mostly used trial and error to select the features for the model. The features I selected were BLEU score, overlap, cosine similarity, levenshtein distance, length difference, meteor score, and NIST score. Using this combination of features, I observed the highest accuracy on the dev set.

ii.) Data Preprocessing and Feature Preprocessing
Methods used to preprocess the data before extracting the features, including preprocessing methods not used in the final model.

Remove spaces before punctuation: Because I planned to expand all of the contractions and remove the punctuation, I had to fix the spacing because there was an additional space before any punctuation. I used regex to remove all of these spaces.
Expand contractions: I expanded all of the contractions using contractions.fix. Because a contraction and the expanded version have the same meaning, expanding the contractions can improve results when finding common words or jaccard similarity.
Lowercase: Strings are case-sensitive, so it makes sense to change all of the letters to lowercase, as this does not change the meaning of the sentence.
Remove punctuation: In most cases, punctuation does not change the meaning of the sentences, so it makes sense to remove the punctuation.
Remove digits: From doing research on the best preprocessing methods, I found that some sources stated that removing digits from the sentences can improve accuracy. But, I found that this significantly lowered the accuracy, so I did not use this preprocessing method in the final model.
Remove stopwords: Stop words are filler words that do not change the meaning of the sentence, so it initially made sense to me to remove these. However, removing the stop words significantly decreased the accuracy of the model, so I kept all of the stop words.
Lemmatization: Lemmatization changes any kind of word to its base root model. This can make it easier to find synonyms. I observed better results with lemmatization rather than stemming.
Stemming: Stemming reduces a word to its word stem. I experimented with stemming, but did not use it in the final model, as I found better results by lemmatizing.
Remove white space: I removed all additional white space that was not needed.
Split: Because most features I was planning on using required lists of words as input, I split all of the sentences by white space, resulting in lists of words.

Similar to features, I also used trial and error to determine which preprocessing methods to use. I found that I achieved better results using more simple methods of preprocessing. Otherwise, preprocessing can remove too much information from the sentences. In the final model, the preprocessing I used included removing spaces before punctuation, expanding contractions, converting to lowercase, removing punctuation, lemmatizing, removing white spaces, and splitting.

The previous preprocessing methods were the same as the ones used in the midterm, but this time, the data given was imbalanced, meaning that there were more negative samples than positive samples. Because this data was imbalanced, I used SMOTE and fit.resample to resample the data to get a balanced set to train on.

iii.) Algorithms and Libraries
pandas:

- I used pandas to read in the csv files and store the data in data frames.

nltk:

- stopwords: used to remove stop words when needed. I did not end up using this in the final model
- WordNetLemmatizer: used during preprocessing to lemmatize the inputs.
- PorterStemmer: I experimented with stemming during preprocessing, but did not end up using this in the final model.

sklearn:

- MLPClassifier(): from the neural network sklearn library, used to fit the data to multi-layer perceptron.

contractions: Used to remove contractions when preprocessing the data.
regex: Used during preprocessing to easily remove all punctuation and white space.
string: Used to easily detect punctuation so I could remove it during preprocessing.
pylev: Used the pylev library to calculate the levenshtein distance.
warnings: Used to suppress the warnings from calculating the BLEU score.
Imblearn: Used to balance the dataset.
PyTorch: Initially, I used the multi-layer perceptron from PyTorch, but I observed better results using the sklearn MLP classifier.

iv.) Experience and Lessons

Overall, I learned a lot about deep learning and neural networks as a result of this project. Prior to starting the project, I read a lot of papers about multi-layer perceptrons and their applications in NLP and detecting paraphrasing. I used this information to develop my own model. Similar to the midterm project, I used a lot of trial and error to test which features worked best and what values to set the hyperparameters to. I tried developing new features and changing which features were incorporated into the model, but I still observed the best results using the same features from the midterm project.

Differing from the midterm project, the data given for the final project was imbalanced. So, I decided to resample the data to get a better ratio of positive to negative samples. Because the sampling is random, I observed a different F1 and accuracy on the dev set each time, but the results were still significantly higher than using the imbalanced data to train the model.

Additionally, this project allowed me to see the value of using F1 over accuracy when evaluating the model. Accuracy is used when the true positives and true negatives are more important, while F1 is used when the false positives and false negatives are more important. So, F1-score can be a better evaluation metric when there is an imbalanced class distribution.