***NLP Assignment #1***
***Preyansh Kaushik***
***McGill ID: 260790402***

**Problem Setup:** To form training and testing data, I decided to treat each line in the dataset as a data point, put these into an array, randomized the order, and then manually pick a split between training and test. For purposes of this experiment, I kept the training-test data split at 80% constant throughout different models. For each sentence, I cleaned the data with different preprocessing tasks. Using nltk's internal libraries, I removed all stopwords from the text, tokenized the sentence, and implemented custom preprocessors that stem using the Porter Stemmer and lemmatize using the WordNet Lemmatizer. For every data point (sentence), a feature vector was created using the CountVectorizer library from scikit-learn. This creates a (#sentences * #features) matrix with unigram counts as the contents of the matrix. Testing data was also preprocessed using the same process, but then transformed to fit within the same features generated from training.

This preprocessing was generally sufficient for all models, apart from the Naive-Bayes Classifier, which required data manipulation as the input format is different.

**Experimental Procedure:** The experiment was run on four different models and a random baseline. The four models are the Naive-Bayes, Logistic Regression, SVM with Linear Kernel, and additionally the Perceptron classifier (a 1-NN linear classifier). On each run, the training and test data is read from the files, randomized, split 80%, fed into the five different classifiers, and then evaluation metrics are obtained with test data.

**Range of parameter settings:** I decided to keep the train-test split, and elimination of stopwords constant throughout the models. There are two variables that change between the models in this experiment . The first is the "min_df" which essentially removes infrequently occurring words as features (as suggested in the assignment). i.e. min_df = 2 will remove words that do not occur in at least 2 different data points. The second parameter is the preprocessing step for the features, i.e. between using a lemmatizer or a stemmer. All combinations of min_df and choosing whether to lemmatize/stem were tried on the five classifiers, and results are tabulated.

**Results:** The following tabulates performance of each classifier with different parameter settings in terms of accuracy. i.e. (# correct predictions / sample size )

*Note: Each category was ran thrice on the models and the average is tabulated*

| Classifier | Min_df = 2 | | Min_df = 4 | |
|---|---|---|---|---|
| | Lemmatize | Stem | Lemmatize | Stem |
| Naive-Bayes | 0.774 | 0.749 | 0.753 | 0.764 |
| Logistic Regression | 0.753 | 0.741 | 0.742 | 0.741 |
| Linear SVM | 0.741 | 0.722 | 0.717 | 0.716 |
| Perceptron | 0.726 | 0.701 | 0.700 | 0.689 |
| Random Baseline | 0.502 | 0.518 | 0.506 | 0.489 |

**Conclusion:** From the results, it seems that the Naive-Bayes classifier performs the best across all other models. There is generally stronger performance along lemmatizing rather than stemming the words, and the difference between removing infrequent words between 2 and 4 is almost negligible. The performance with lemmatizer is particularly stronger, perhaps because more words map to the same lemmatized from rather than with stemming. This helps keep features more consistent across training and test data. It is important to note that the model is trained with different data between each parameter setting because of random split of data, and so the feature vector unigrams could be of different shape every time. For purposes of comparison, however, this experiment suffices since all the models are run consequently on same parameters.

**Confusion Matrix for Naive-Bayes:** Since Naive-Bayes performed the best among the other classifiers, metrics for a confusion matrix on its best performance parameters (min_df = 2 & lemmatize) were obtained:

| *n = 2133* | Actual Positive | Actual Negative | |
|---|---|---|---|
| Predicted Positive | True Positive = 830 | False Positive = 228 | 1058 |
| Predicted Negative | False Negative = 247 | True Negative = 828 | 1075 |
| | 1077 | 1056 | |

From the confusion matrix analysis, the accuracy is at 0.778 for this particular test run. However, the false discovery rate for the model is still high at 0.215, yielding this a medium performance model. In the future, tuning train-test split and increasing the range of the min_df parameter could perhaps yield better results.