Text Classification Experimental Analysis

Claudia Porto

In this project our aim was to build a text classifier that is able to distinguish between words that are simple and words that are complex. Text classification is one of the first steps in a NLP task called text simplification, which makes text easier to read by replacing complex words with simpler ones, etc. Whether it be younger readers, people with learning disabilities, etc, this NLP task of text classification makes it easier for people to read more high-level texts.

Baseline Classifiers:

I began by implementing three simple baselines as classifiers and evaluating their performance on both the given training file and development file.

All Complex Baseline Classifier

The first baseline classifier implemented was a very simple one: it labels all words in the given data file as complex. The table below shows the performance of this classifier on both the training data and the development data.

Training Data Performance:

Accuracy	43%
Precision	43%
Recall	100%
F-score	60%

Development Data Performance:

Accuracy	44%
Precision	44%
Recall	95%
F-score	61%

This is a very simple classifier, so it was expected for it to not perform very well. We see that the all_complex baseline classifier performs slightly better on the Development Data than the training data. This could be due to the number of words in the file (4000 in the training file compared to 800 in the development file). The recall is the only evaluation metric that had no change from one file to the other, staying at a constant 100%. It is also interesting to note that the accuracy and precision are equal to each other.

Word Length Baseline Classifier

The second baseline classifier implemented was a slightly more complex one: it uses the length of each word to predict its complexity. This is accomplished by using the f-score to find the best length threshold and using this threshold to classify the data. The table below shows the performance of this classifier on both the training data and the development data for the best threshold that I found to be 7.

Best Threshold: 7

Training Data Performance:

Accuracy	69%
Precision	60%
Recall	84%
F-score	70%

Development Data Performance:

Accuracy	70%
Precision	62%
Recall	86%
F-score	72%

In comparison to the all_complex baseline classifier, this word length baseline classifier performs much better. Similar to the all_complex baseline classifier, this classifier performs slightly better on the development data than the training data, even though we used the same best threshold on both files. Again, this could be due to the number of words in the file (4000 in the training file compared to 800 in the development file).

Word Frequency Baseline Classifier

The third baseline classifier implemented is similar to the word length classifier: it uses the frequency of each word to predict its complexity. This is accomplished by using the f-score to find the best threshold and using this threshold to classify the data. The table below shows the performance of this classifier on both the training data and the development data for the best threshold that I found to be 1883.

Best Threshold: 1883

Training Data Performance:

Accuracy	44%
Precision	43%
Recall	96%
F-score	60%

Development Data Performance:

Accuracy	44%
Precision	44%
Recall	95%
F-score	60%

The minimum frequency was 40 and the maximum frequency was 47,376, 829,651. Because the maximum frequency was such a massive number, I settled upon testing the thresholds from 40 to 25,000 to identify the best threshold for the model. In doing this, I found that the best threshold happened to be 1883. In comparison to the word length baseline classifier, this word frequency baseline classifier performs worse. The performance on the training data and development data stays relatively the same, other than a 1% increase in precision and a 1% decrease in recall for the development data. This is interesting because in each baseline classifier thus far, we have seen an increase in the performance from the training to the development data. In fact, the performance results we see here are very similar to those of the all complex baseline classifier that we previously implemented. The accuracy, precision, and f-score are all relatively the same, the only slight difference are the recall values (100% for all_complex and 96% for word frequency).

Classifiers:

Next, I looked at different types of machine learning classifiers using built-in models from sklearn, to train a classifier. The features that were included were normalized according to the training data.

Naive Bayes Classifier

For the first classifier, I used the built-in Naive Bayes model from sklearn. The features used were word length and word frequency, according to the Google ngrams. The table below shows the performance of this classifier on both the training data and the development data.

Training Data Performance:

Accuracy	55%
Precision	49%
Recall	98%
F-score	65%

Development Data Performance:

Accuracy	55%
Precision	50%
Recall	98%
F-score	66%

The Naive Bayes classifier performed relatively well, with a high recall value, meaning that it has a low rate of false negatives. From the training data to the development data, there is a slight increase in both precision and f-score, however, the accuracy and recall remained equal in both data sets.

Logistic Regression Classifier

For the next classifier, I used the built-in Logistic Regression from sklearn. The features we used were word length and word frequency, according to the Google ngrams. The table below shows the performance of this classifier on both the training data and the development data.

Training Data Performance:

Accuracy	74%
Precision	72%
Recall	65%
F-score	68%

Development Data Performance:

Accuracy	77%
Precision	76%
Recall	70%
F-score	73%

The Logistic Regression classifier performed much better than the Naive Bayes classifier, according to the accuracy and precision evaluation metrics. However, the recall value was much lower than the Naive Bayes classifier, meaning that it has a higher rate of false negatives. From the training data to the development data, we see a significant increase in all evaluation metrics.

Build Your Own Classifier

For our last classifier, I experimented with the built-in SVM, Decision Trees, and Random Forest classifiers from sklearn. The features I tried were the number of synonyms, and syllable count, according to WordNet synonym counts.

SVM Classifier:

The first classifier I experimented with was the Support Vector Machines (SVM) classifier. SVM is a popular algorithm for classification tasks. It uses a hyperplane to separate the data into classes, with the goal of maximizing the margin between the two classes. The table below shows the performance of this classifier on both the training data and the development data.

Training Data Performance:

Accuracy	72%
Precision	70%
Recall	59%
F-score	64%

Development Data Performance:

Accuracy	75%
Precision	75%
Recall	64%
F-score	69%

Decision Tree Classifier:

The next classifier I experimented with was the Decision Tree classifier. Decision trees are a simple machine learning model that can be used for classification tasks. They work by recursively splitting the data based on the most significant features, until they reach a leaf node that represents a prediction. The resulting tree can be used to classify new instances by traversing the tree from the root to a leaf and returning the associated class label. The tree can also be visualized to understand the decisions made by the model and the relative importance of each feature. The table below shows the performance of this classifier on both the training data and the development data.

Training Data Performance:

Accuracy	73%
Precision	70%
Recall	64%
F-score	67%

Development Data Performance:

Accuracy	73%
Precision	70%
Recall	64%
F-score	67%

Random Forest Classifier:

The final classifier I experimented with was the Random Forest classifier. Random forests combine multiple decision trees to create a more accurate model. In a random forest, many decision trees are trained on different random subsets of the data and features, and the final prediction is made by aggregating the predictions of all trees. The table below shows the performance of this classifier on both the training data and the development data.

Training Data Performance:

Accuracy	73%
Precision	70%
Recall	64%
F-score	67%

Development Data Performance:

Accuracy	73%
Precision	71%
Recall	66%
F-score	68%

Although these three different classifiers' performance values all fell around the 70% range, I found that the SVM classifier performed the best overall on the development data. I chose the SVM classifier as my best model. I found it interesting that the Decision Tree classifier performed the same for both the training data and development data. Also, the Decision Tree classifier and the Random Forest classifier performed about the same, except for a slight increase in precision, recall, and f-score for the Random Forest classifier. The slight increase in performance is expected because the Random Forest classifier uses Decision Trees within the model.

Using the SVM classifier, I found that my model is making errors on words that include hyphens, and words with double letters. Hyphenated words that were incorrectly classified include but are not limited to "fast-moving", "same-sex", "long-buried", "best-selling", "spider-man", "ma-ah-kay", "near-vertical", "grayish-white", "closed-door", "north-central", "ag-minded", "no-food-or-drink", and "go-ahead". Words with double letters include "official", "refugees", "blooms", "commissioner", "anniversary", "accepting", "ballast", "attorney", "footage", "affirmed", "fiancee", "amassed", "proceeded", and many more. These are just two of the many different

possible category types for the words this model performs poorly on. On the other hand, some words that my model is correctly identity include “coaches”, “purchasers”, “authorizing”, “epidemic”, “apartment”, “evaluated”, “acuity”, “fictional”, “prostitution”, “substantial”, “coalition”, “representatives”, etc. This list includes a few hyphenated and double letter words, but a majority of words that fall into those categories fall within the wrongly classified lists.