## Pre-processing

### Representation of Words

I attempted representing words of the abstracts in two primary forms: TFID and word frequency.

* 1. Word Frequency

Word frequency was the representation that I ended up using for my final prediction. This created an array of all words seen in the training data, and counted the frequency that word was seen in each particular abstract. When looking at validation data (in k-fold, or in the tst.csv), only those words that have been previously seen are considered, with the other ones skipped. This provides a matrix of frequencies which can then be passed to the later models.

* 1. TFIDF

The Term-frequency, Inverse-Document-frequency representation is a weighted representation that takes the count of words as above (term frequency) but multiplies by the inverse frequency of that particular word across all the documents seen in training. This effectively weights each particular word for its uniqueness in the dataset, meaning it’s ability to differentiate between different documents.

### Data Processing

In addition to the above word representation, a pearsons correlation was created to see if it could help in the selection of the k-best features. Though using a correlation should in theory lead to better results, in practice, selecting the words with the greatest frequency between documents yielded better results, and so this was chosen for the final prediction.

## Naïve Bayes Implementation

The implementation of Naïve Bayes follows that of the general theory. The model is fitted by generating priors, and probabilities for each word seen in the training data. Priors were equated to the number of instances seen from the class divided by the total number of instances. Probabilities for each class were calculated as the frequency of a particular word in a particular class divided by the total number of words (frequency included, non-unique) in a particular class.

The log function was used to add probabilities (same effect and order preserved in log addition as in multiplication) in order to prevent underflow, when multiplying very small probabilities. The predicted class was still that with the greatest value, as a smaller likelihood becomes and increasingly negative number (order preservation).

Prediction simply took the log probability of seeing each word in a particular class, added to the priors for that particular class, with the maximum returned as the predicted class.

## Naïve Bayes Extensions

|  |  |
| --- | --- |
| Class | Number of Examples from Class |
| A | 128 |
| B | 1602 |
| E | 2144 |
| V | 126 |

For the extension, I implemented the Naïve Bayes Complement classifier. According to SKLearn “The Complement Naive Bayes classifier was designed to correct the “severe assumptions” made by the standard Multinomial Naive Bayes classifier. It is particularly suited for imbalanced data sets.”. The dataset provide was relatively unbalanced (see table above) with a few thousand instances from two classes, and only a few hundred for the other two. The aforementioned imbalance results in the standard naïve bayes selecting poor weights for the decision boundary; the weights of these minority classes are shrunk. Additionally, the standard naïve bayes assumes independence of words, even when they are in fact (bound by natural language) inherently dependent. Using the complement class mitigates this issue by combing all other classes into the ‘complement’ class, and taking the weighting of this class, rather than one class at a time.[[1]](#footnote-1) Essentially, the class that is **least likely** to **not contain** the word is the one that is predicted by the Naïve Bayes Complement Classifier.

The complement naïve bayes classifier was specifically chosen due to the advantages it presents on skewed datasets, such as the training one provided (see table above showing imbalance). Complement performs well in this scenario compared to the vanilla naïve bayes, as well as other extensions including multinomial, which suffers from predicting based off one class as describe above.

## Performance Evaluation Methodology:

### KFold

To ensure a fair representation of model performance, K-fold Cross Validation was used. This gives a better estimate for model accuracy by evaluating over the whole training set multiple times. The K chosen for this purpose was 10; large enough for good accuracy, whilst being time conscious.

### Stratified

To account for the underrepresentation of some classes in the dataset, the aforementioned k-fold cross validation was also stratified. This ensured that there were representative percentages of each class within every fold, rather than some folds missing classes entirely as would be possible with the distribution of the dataset.

## Model Performance (☹)

### Standard

The standard Naïve Bayes classifier had a training accuracy of ~3% when using tfidf, and an accuracy of **INSERT ACCURACY.** This was a reasonable accuracy, but is slightly less than could be expected from the dataset (~90%). Despite **lots** of testing, I couldn’t seem to work out why ☹.

The difference between these two methods can potentially be explained by the number of features. In tf-idf, no feature selection was used, unlike select-k-best for word frequency. The extra dimensionality of the tf-idf version of the data could have mislead the model, becoming overly biased towards each training fold, and not generalising very well.

### Extended

The extended naive bayes classifier had a training accuracy of **INSERT ACCURACY** percent when using tf-idf. This is definitely the result of implementation, but again, unfortunately, despite **many** hours, I could not determine the cause.

The model had an accuracy of **INSERT ACCURACY** when using the word frequency. This drop can only be explained by an incorrect implementation; there should have been improvement for the reasons described above.

1. Source: http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf [↑](#footnote-ref-1)