Assignment 2

1. Environment Set Up
2. Pre-processing and Feature Selection

This section outlines the pre-processing steps taken and then what additional activities were performed to select the relevant features. Python was used here because the range of libraries available for feature extraction of text are, in my opinion, better and easier to use. Specifically the use of the SpaCy library makes it easier to extract lots of information from one pass of the data.

* 1. Data Cleansing

The raw data files are loaded into a pandas dataframe ensuring that ID, TEXT and EOD are used as delimiters and line breaks. Text is passed to the Spacy interpreter where it assigns a series of attributes to each token from a pre-trained and very accurate series of algorithms. From here I have completed the following cleansing activities using the attributes given. They are:

* Remove punctuation
* Remove excess whitespace
* Remove URL like text
* Remove email address or like
* Remove numbers or like numbers
* Remove currency
* Remove quotation marks
* Lowercase everything, excepted recognised entities (discussed later)
* tokenisation

This level of cleaning means the text is free from elements likely to create noise in the document and is rarely, if ever, useful for understanding the meaning of text. This provides a good set of tokens.

* 1. Feature Selection

Possibly the most important part of the pre-processing step is deciding how much to enrich features beyond a simple bag of words model. Given the size of the training corpus and the large number of classes, adding in too many variations on each word through enhanced features could reduce the occurrence of key terms too much to make useful predictions without even more samples. Two methods were used to generate features and they’re compared below.

|  |  |  |
| --- | --- | --- |
| Method | Activities | Evaluation |
| GloVe word embeddings | * Average word embedding for each word after cleaning (300 feature vector) | Using a pre-trained embedding can produce better results but in this case performance was significantly worse than TF-IDF with POS tagging and was discarded |
| TF-IDF POS Tagged | * Add POS tagging to each word * Take the lemma of each word * Allow named entities to remain as they are * Use unigram and bigram representations (with POS tags) * Remove any word unigram or bigram appearing in more than 98% of docs * Transformation to TF-IDF with L1 regularisation to remove unnecessary features | TF-IDF vectors performed quite well and produced a good set of features.  Lemmatized and part of speech tagged words created a larger feature vector after L1 normalisation of >1500. This has computational impacts. |

The end result of TF-IDF is a sparse matrix representation of a bag of words with some additional morphological attributes maintained to improve prediction accuracy and some features taken out of the model to help with size.

Other methods considered but not implemented:

|  |  |
| --- | --- |
| Method | Reason |
| Grid search to tune parameters of TFIDF and Count Vectorizer fit to a simple linear classification algorithm with CV for comparison | Given the size of the training data (106k) and the possible hyperparameters, this task becomes too computationally inefficient to complete in a reasonable time |
| Take a small sample of the training data and then run the grid search above | Any sub-sampling comes with possible bias introduced from the selection. It also would need to be a very small sample (maybe a few thousand) to make it possible to run various combinations |
| Use Word2Vec representation of word structure to create feature vectors where word meaning and context are used across documents instead of actual words | Will not necessarily perform better than GloVe vectors. Training a word embedding for just this task may also not improve results |

3 Model Selection

In this section we discuss the process used to select and tune the right model to minimise the macro F1 score. We’ll use a grid search approach with cross folds validation to help narrow down the search.

3.1 Model Assessment

To assess the correct model, 5 possible ML models that are recognised as good multi-level classifiers were chosen to be evaluated. Each model was evaluated with the following:

* 30,000 randomly selected data
* 5 folds cross validation
* Macro F1 score error term
* A grid-search of the hyper-parameters for these models

The results are below:

|  |  |  |
| --- | --- | --- |
| Model | Parameters Tuned | Best Error |
| SVM | Cost (0.01,0.1,1,10,100,1000)  Gamma(0.0001,0.001,0.01,0.1,1,10) |  |
| Rpart | Minsplit(20,30,50)  Cp(0.01,0.1,0.2) |  |
| Random Forest | Mtry(20,39,78)  Ntrees(1000,3000) |  |
| K-Nearest Neighbours | K(3,5,7,11) |  |

4 Model Implementation

Running any deeper grid-searches will take too much time, even with the ML techniques narrowed down to the best 1 or 2. Therefore we will build our model off the best performing 30,000 grid searched parameters.

The model is implemented in the file assignment2\_r.r. It takes the feature vectors of the training and test sets and outputs the testing\_labels\_final.text file