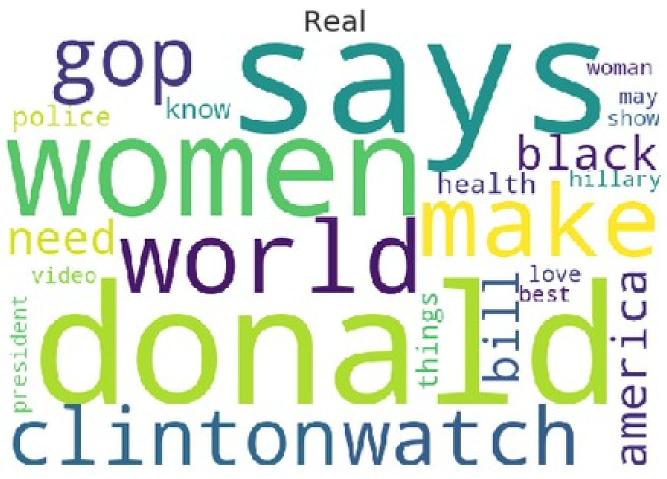
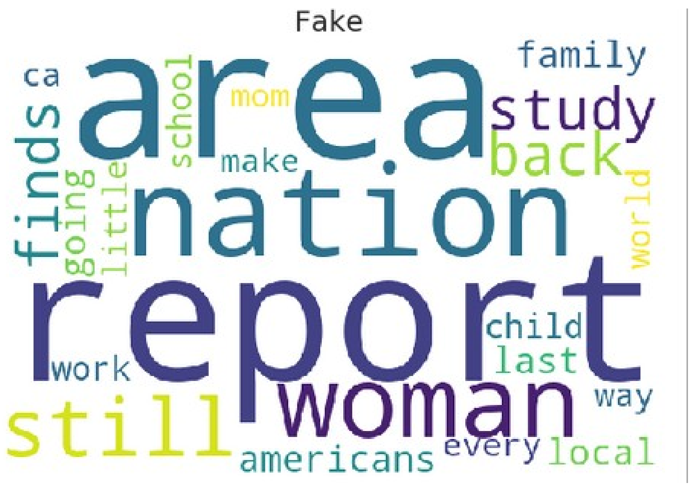
Fake News Classifier

EE514 Assignment
by
Liam Barry

No: 19215629





Pre-Processing & Splitting

This dataset was comprised of 28619 newspaper headlines from a variety of sources labelled as real or fake news. Code sections in the accompanying Jupyter notebook are referred to throughout beginning with Section 1.1 & 2 which correspond to cleaning & processing. Firstly, the dataset was loaded into a pandas dataframe and the source column of the articles discarded despite potential value as a classifier. This was to ensure robustness of the model to idenitfy fake news based on headline alone. Preliminary EDA & cleaning confirmed no missing values or major imbalance in the dataset (n = 14985 for 'Real', n = 13634). The ‘raw’ dataset also appeared to have undergone some processing as all examined headlines consisted of lowercase characters. The value counts of the two categories were visualised using a bar chart (Figure 1). It was also found that the data labels were stored as integers rather than categories and this was amended using the ‘as.type()’ function.

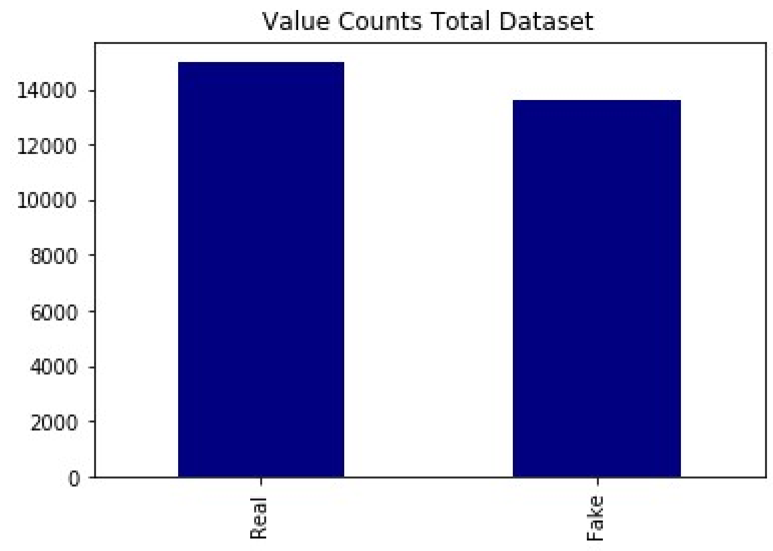


Figure 1: Value counts of the number of headlines in each category.

Examination of the dataset in Section 1.3 uncovered several duplicate headlines; some with up to 12 copies, and these were removed using the pandas drop.duplicates function; keeping just one copy of each. This was considered the best approach despite some of the headlines appearing to correspond to weekly publications e.g. ‘sunday roundup’ and ‘the funniest tweets from parents this week’. While duplication affected both datasets it’s was particularly prevalent in the ‘fake’ set with a total of 150 duplicates removed versus 46 for the real set. Unfortunately the extent of duplication in the dataset was not discovered until model building & evaluation had been carried out and this presents a major limitation of the report; particularly as filtering duplicates would have exacerbated the class imbalance and has significant potential to affect performance as word frequency was the primary feature used in training each classifier. Figures presented hereafter for he dataset include duplicates and this should be borne in mind when interpreting conclusions. The code for removing duplicates is included in Section 1.1 but is commented out to allow results to be reproduced.

The dataset was shuffled (with random seed = 1) and split into a training/test set in the ratio of 75:25 using sklearn (Section 1.2). The training set was further divided to create a validaton set using 10% of the training samples. Maintenance of the class balance was ensured at each split using the ‘stratify’ parameter in the sklearn splitting function and the success of the splits verified through value counts (Split sizes: n = 19316 for Training, n = 2147 for Validation and n = 7155 for Test). Although sklearn has the functionality to balance the dataset (50:50 Real:Fake news) through re-sampling or discarding samples from the majority class; neither approach was deemed necessary at this stage given the possibility that the inherent ratio might be more representative of the real world.

Exploratory Data Analysis

In Section 2.1, the headlines were tokenized using the NLTK library to allow the distribution of word-counts of the two categories in the train set to be compared using a horizontal boxplot. An extreme outlier was noticed in the 'Fake' data and investigated. This 'headline' appeared to mistakenly include the entire story (word count of 165) but on-checking the source it was confirmed that the headline length was correct. Nevertheless, the potential for this outlier to obscure patterns in the data prompted it's removal & exclusion from analysis (the outlier is removed in Section 1.1). Comparison of word counts in the remaining samples did not reveal any obvious patterns in the data with both sets having highly similar distributions evidenced by their mean & standard deviation(x = 10.71 and σ =3.30 for Real, x = 10.83 and σ = 3.97 for Fake).

The next step was to analyze frequency distributions of common words in the training set. This was carried out using NLTK's 'FreqDist' function. Initially the raw distributions were compared and the 30 most common were predominantly stop words/punctuation as expected (Figure 2). Interestingly, after stop-word removal both sets exhibited marked similarity in their most common words (14 out of 30 most-common words were present in similar frequencies in both categories). Graphs of common word counts before/after stop-word removal were generated (Figure 3a & 3b) and the results intended to be used to eliminate words with highly similar frequencies deemed unlikely to be capable of discriminating real from fake news. After removing the 14 shared words and repeating the frequency distributions just 5 of the top-30 words were shared. The frequency distributions of common words post stop-word removal were also used in conjunction with the wordcloud package to generate the graphics on the report cover.

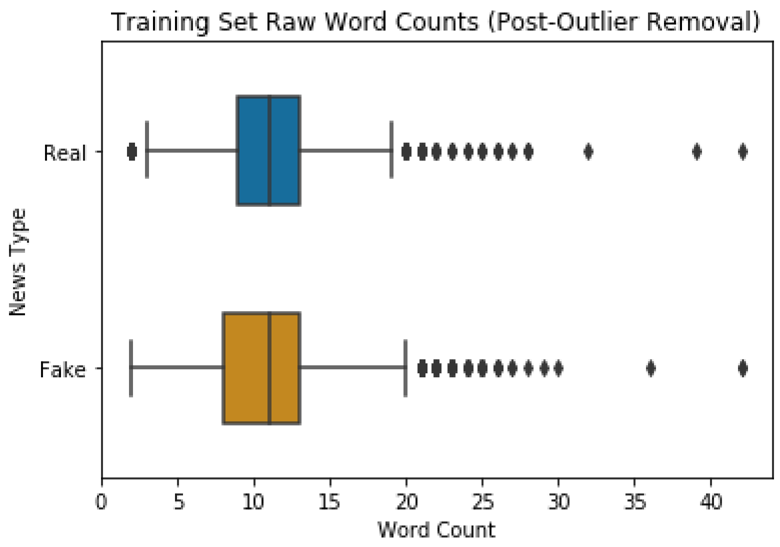
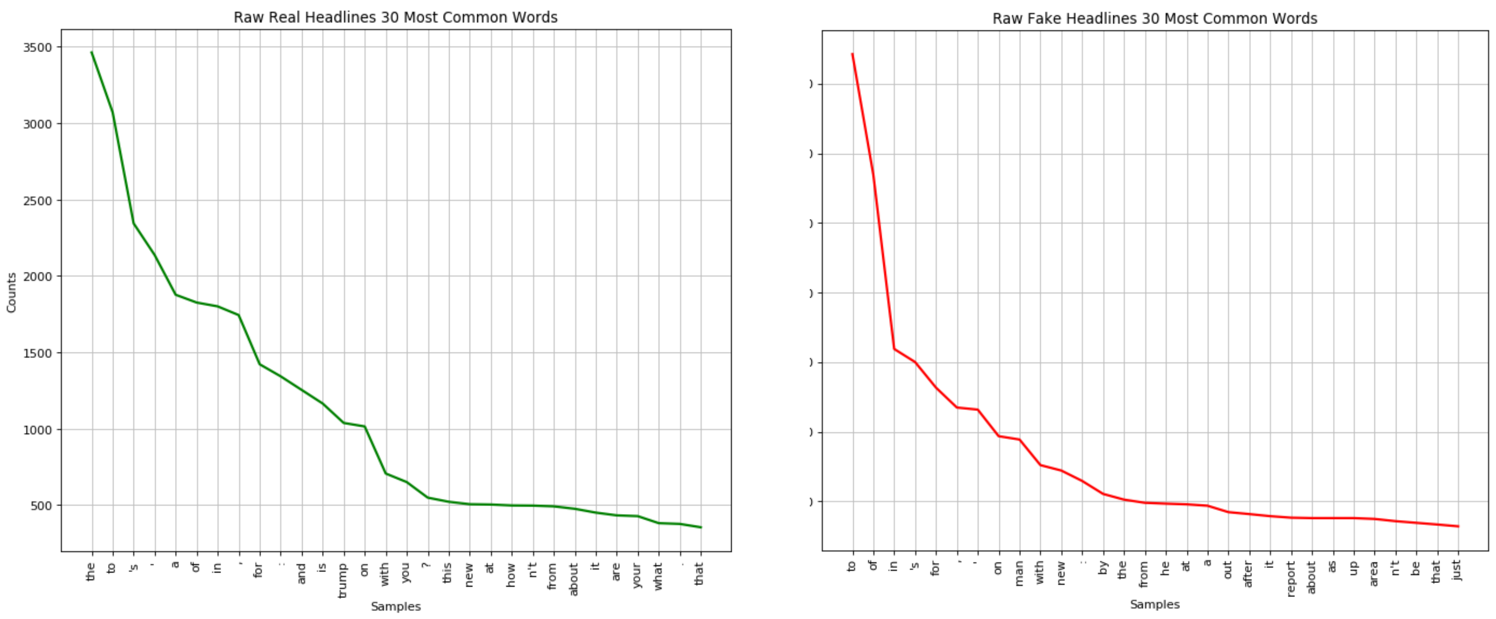


Figure 2: Boxplot of word counts for Real & Fake emails in the training set



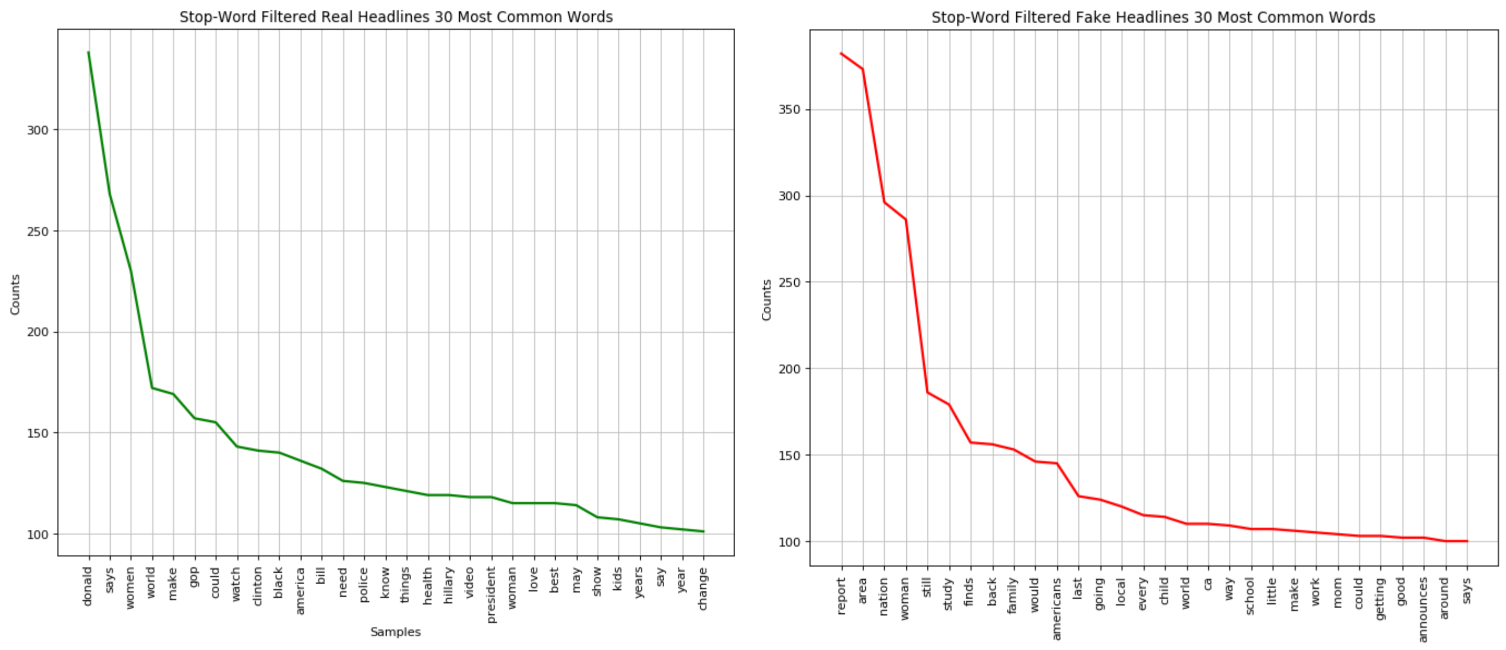


Figure 3a(above) & 3b (below). Frequency counts of most common words in Real (green) and Fake (red) headlines in the train set. Please see jupyter notebook for enlarged images and/or code for viewing frequency counts in table form.Table 1 in the notebook shows the common words post stop-word removal.

However; it emerged during model selection that both stop-word and shared common-word removal had a detrimental effect on model performance without a noticeable improvement in computation speed (discussed below). The absence of sensationalist language in the fake news set was also surprising; as was the similarity in most common words. This was seen as indicative of the level of sophistication of fake news when compared to the typically easily identifiable fake email/SPAM. A side-effect of the tokenization procedure also caused some distortion of the word counts as the suffix " n't " for example was removed. The common-words list was examined for words that could have been affected by this procedure (could, would, should etc) but no candidate words were present in both sets so this was not investigated further. Removal of punctuation, particularly quotation, question and exclamation marks was also noted as being potentially detrimental to model accuracy and an avenue for improving performance.

Feature Extraction

Initial feature extraction consisted of a simple unigram Bag of Words approach implemented with ‘count_vectorizer’ in sklearn which involves creating an encoded matrix of all words in the dataset and comparing their frequencies in the two categories. This simplistic model can be tuned through changing to a bigram, trigram etc model where 2 or 3 words in sequence are stored as opposted to single words in the unigram approach. Performance was evaluated by calculating the training & validation accuracy in order to gain an insight into the impact of different word-removal techniques as BoW's with large vocabularies were thought to be computationally expensive while smaller ones may be more prone to over-fitting. Note that as the dataset was slightly imbalanced; the proportion of real news (54%) should be considererd the benchmark for evaluating a model i.e. an accuracy of at least 54% would be expected if a model simply learns the ratio in the training set.

As a baseline, the unigram model was first fit to the raw training set using a Multinomial Naive Bayes model in Section 3.1 and acheived validation & training accuracy of 84.96% & 93.14% respectively. This was then compared with performance post stop-word removal and a slight decrease in accuracy was observed (V=79.88%, T=91.33%) and a further (but far less significant) decrease when common-shared words were removved (V=79.51%, T=91.42%). This was initially surprising but on consideration it should be expected as removing parameters will invariably lead to a decrease in accuracy on the training set. What was striking was the large drop in validation accuracy post-stop word removal. Investigating the size of the BoW matrices for each approach revealed the size of the vocabulary was not significantly affected by stop-word removal (22157 elements vs 21868 with shared & common English stopwords removed). The size of the matrices was computed in Section 3.1.Sidenote.

In the Bag of Words (which is also known as Term Frequency) model, the overall frequency of words occurring in each class is counted but the number of seperate headlines containing each word is not assessed. TF-IDF(Inverse Document Frequency) attempts to measure the importance of a given word across the class by calculating the logarithmically scaled inverse ratio of the number of headlines that contain the word and the total number of headlines as per equation 1:

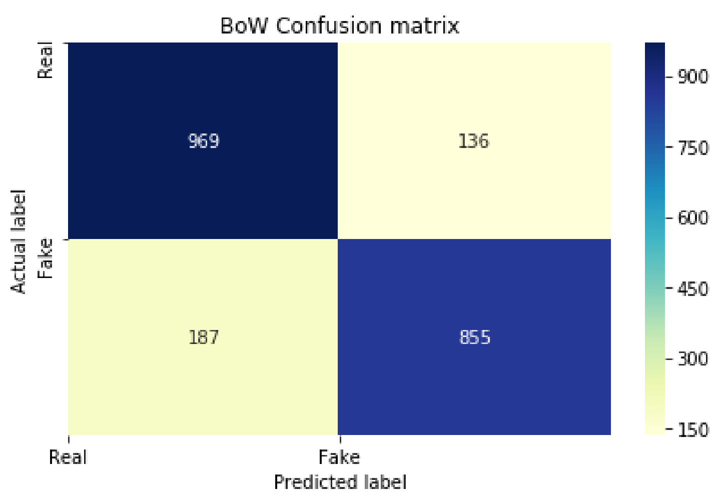
Equation 1 $$\mathrm{idf}(\mathrm{W}) = \log \frac{\#(\text{documents})}{\#(\text{documents containing word W})}$$

TF-IDF normalizes the document term matrix. It is the product of TF and IDF. While this model is more sophisticated, it resulted in poorer prediction accuracy than the Term Frequency alone; with validation & training accuracy of 82.99% and 92.56% respectively for the raw headlines. Summaries of the performance of each feature extraction approach with Multinomial Naive Bayes are available in notebook section 3.1 and 3.2 but it was apparent that stop-word removal was detrimental to model performance for both feature extraction approaches and model selection was carried out on the raw headlines without stop-word removal. While bi- and tri-gram approaches were evaluated; both showed significant over-fitting with training accuracy approaching 100% without any improvement in training accuracy despite heavier computation load.

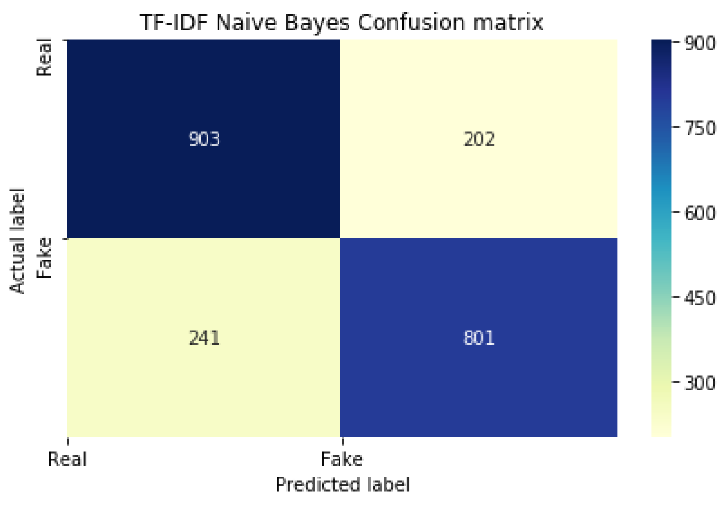
Model Selection

Results of early model evaluation were corroborated by similar news classifiers described online; namely that stop-word removal had a noticeable negative impact on accuracy, bigram (and higher) BoW/TF-IDF approaches were highly susceptible to overfitting and the computational savings gained from reducing vocabulary size were not outweighed by the decrease in model performance. Initial experiments indicated the optimum approach was to use a Bag of Words/count vectorizer as a feature extracter for Multinomial Naive Bayes but this was confirmed by performing a small scale hyper-parameter grid search and using logistic regression in place of Naive Bayes.

Detailed comparison of the default BoW and TF-IDF approaches in conjunction with Naive Bayes is shown in Figure 4 using confusion matrices. As this dataset was approximately balanced and there was no clearly more desirable outcome between misclassifying real news as fake versus the reverse; accuracy and misclassification rate were deemed the most pertinent metrics for comparing models. The BoW appraoch outperformed TF-IDF in every metric except precision when evaluated on the validation set and as such had a substantially lower misclassification rate of 14.29% (vs 19.7%).



```
Accuracy: 84.96 %
Precision: 86.28 %
Recall: 82.05 %
Specificity: 87.69 %
Misclassification Rate: 14.29 %
```



```
Accuracy: 83.0 %
Precision: 87.49 %
Recall: 75.82 %
Specificity: 81.72 %
Misclassification Rate: 19.7 %
```

Hyperparamters represent model factors whose optimal settings are highly dependent on the dataset and cannot be known in advance. A logistic regression model was also fit to both feature sets and had marginally poorer performance metrics to naive Bayes the performance of both models was evaluated in detail by varying hyper-paramters. The code & results of the hyper-parameter searches are covered in detail in Section 4.2 & 4.3 of the accompanying notebook and the rationale explained in the report below.An example of the code used for the hyper-parameter search based on the model_selection.GridSearchCV module from sklearn and it's output is included in Figure 5 for reference. A total of four models were examined representing the different combinations of feature extraction and classifers:

Model 1: BoW and Logistic Regression
Model 2: BoW and Multinomial Naive Bayes
Model 3: TF-IDF and Logistic Regression
Model 4: TF-IDF and Multinomial Naive Bayes

For Multinomial Naive Bayes the most important hyper-parameter is the alpha value used to ensure a non-zero probability is always returned; it's default value is 1. A hyper-parameter grid-search varying the value of alpha between 0.09 and 25 revealed that the optimal value was dependent on the 'n' gram of the bag of words used for feature extraction. While lower values of alpha sporadically resulted in improved performance for the bi- & trigram models they again caused training accuracy to approach 100%; causing concerns about over-fitting. This in combination with the lower memory & computation required for the unigram approach suggested it may be the preferable model. Three fold cross validation was used during the grid-search as this module does not allow for custom validation sets to be used; therefore accuracy values reported for the grid search could not be directly compared with validation accuracy calculated in the previous section but were deemed fit for purpose.

For logistic regression; the most commonly modified hyper-parameter is the penalty applied to the loss function which is generally the squared magnitude of a co-efficient (l2 or l2 norm) or the absolute value of a co-efficient (l1 or l1 norm). Applying these penalty terms is known as regularization and is incorporated to mitigate over-fitting in logisitic regression. The Inverse regularization parameter 'C' has a similar function and was also varied during the grid search. Marked increases in performance were consistently observed for the 'l2' norm compared to the 'l1'. Values of 'C' greater & less than 1 caused marked decreases in performance and the default values for the model were established as the optimum.

For completeness; a hyper-parameter grid search was also carried out for the combination of TF-IDF and Naive Bayes/Logistic Regression. The hyper-parameters of interest for TF-IDF are again the uni-/bi-/tri- gram setting and the minimum & maximum frequency thresholds for including words in the model fitting. When combined with logistic regression the optimal parameters were a unigram approach with max and min thresholds of .75 and 2 respectively; meaning words that occured in ***over 75%*** of headlines or ***less than 2*** headlines were ignored (using proportions of the dataset for the minimum threshold severely affected performance and it is common to set an integer minimum threshold for this parameter rather than a fraction).

The corresponding logistic regression model again had optimal performance when the 'l2' norm penalty was implemented and with 'C' value 2. The variations in performance with changing hyper-parameters were carefully examined for patterns using the 'cv_results' output of GridSearchCV objects and modified before being re-run. For instance, the initial range of alpha values had a maximum of '1' but investigation showed performace increased steadily with alpha which prompted the inclusion of values greater than 1 and the final finding that a value of '3' was optimal. The performance of each of the 4 optimised models

Code Excerpt from Section 4.2 of Notebook

```
#Model 2: CountVectorizer (BoW) combined with Multinomial Naive Bayes
pipe = Pipeline([('cvec', CountVectorizer()),
                 ('nb', MultinomialNB())])

# Tune GridSearchCV
pipe_params = {'cvec__ngram_range': [(1,1)],
               'nb__alpha': [.09,.18,.36,.54,.72,.81,1,1.1,1.25,1.5,2,3,5]}

gs_mnb = GridSearchCV(pipe, param_grid=pipe_params, cv=3)
gs_mnb.fit(X_train, y_train)

#The actual performance of the model during cross-validation when fit with the hyper-paramters
print("Best score:", gs_mnb.best_score_)
print("Train score", gs_mnb.score(X_train, y_train))
print("Test score", gs_mnb.score(X_val, y_val))
```

```
Best score: 0.8366
Train score 0.9142
Test score 0.8467
```

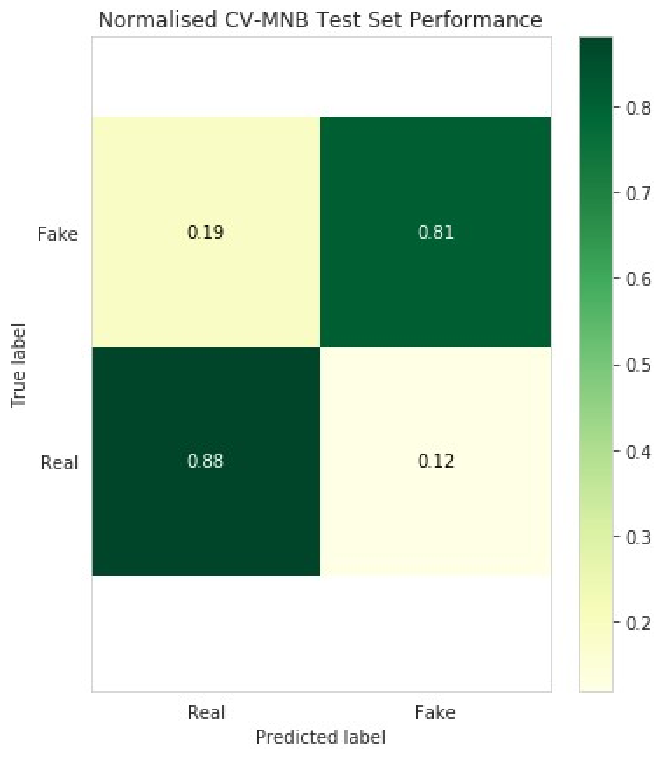
```
#Print out the actual parameter settings that were found to be optimal

gs_mnb.best_params_
```

```
{'cvec__ngram_range': (1, 1), 'nb__alpha': 3}
```

Figure 5: Code for hyper-parameter grid search with scores on train & test included

Model Evaluation & Conclusions

Having established that the count vectorizer combined with Naive Bayes was the superior model it's performance on the test set was evaluated and the types of error the model was making investigated more thoroughly. Normalised & absolute count confusion matrices were generated and the performance metrics visualised using these matrices and an ROC curve. A combination of sklearn's metric module and a small open source library scikitplot were used to generate the graphs. Model performance was impressive with an accuracy of 84.88% on the test set; which was marginally higher than that on the validation set. Similarly; the misclassification rate of 14.08% was also superior to that observed on the validation set but these differnces were not substantial. The normalised confusion matrix allows easy interpretation of the nature & frequency of these errors and showed that the model was particularly adept at correctly identifying real news (12% misclassification vs 19% for correctly identifying fake news).



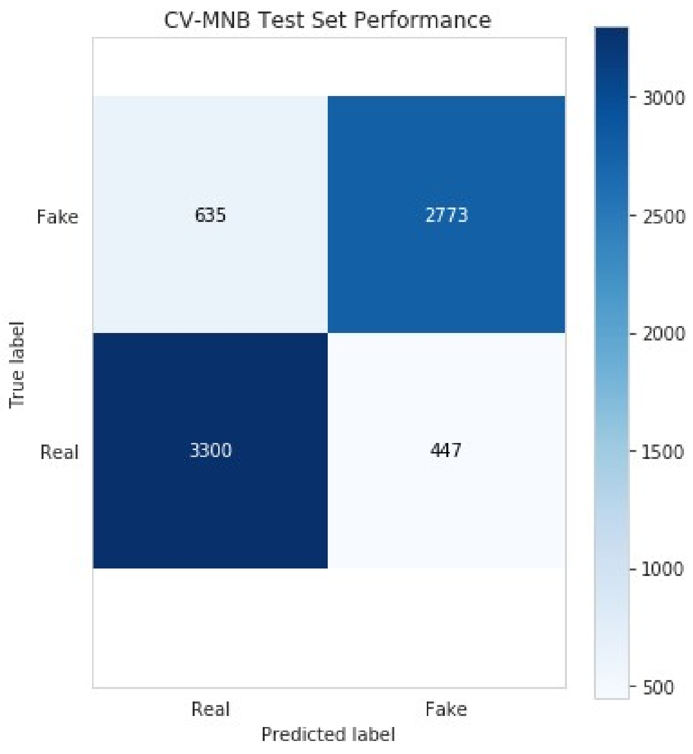


Figure 6: Normalized (left) and absolute count confusion matrices for the final optimized model. Note that 88% of real headlines were correctly identified compared with 81% of fake.

This presents a clear opportunity to boost model performance through downsampling the real news as this was the slight majority class or alternatively tweaking the model to penalise more heavily for misclassification of fake news. As previously mentioned, due to the imbalance in the dataset a no-information or baseline classification rate of 54% should be considered the benchmark for model performance; rather than 50% given that a model will tend to simply learn the class distribution during training.

The relatively poor performance of the classifier in identifying fake news was likely exacerbated by the level of duplication in this subset of the data and repeating the analysis outlined here with a balanced and properly processed dataset would likely improve performance substantially. The presence of duplicates may also explain the poorer performance of the more sophisticated TF-IDF approach as this considers the number of documents containing each word. This oversight was particularly unfortunate given the small relative differences between models as removal of duplicates could significantly affect performance on unseen data. Nevertheless, the final model had impressive performance metrics and an area under the curve (AUC) of .9252. The corresponding ROC curve is shown in Figure 7

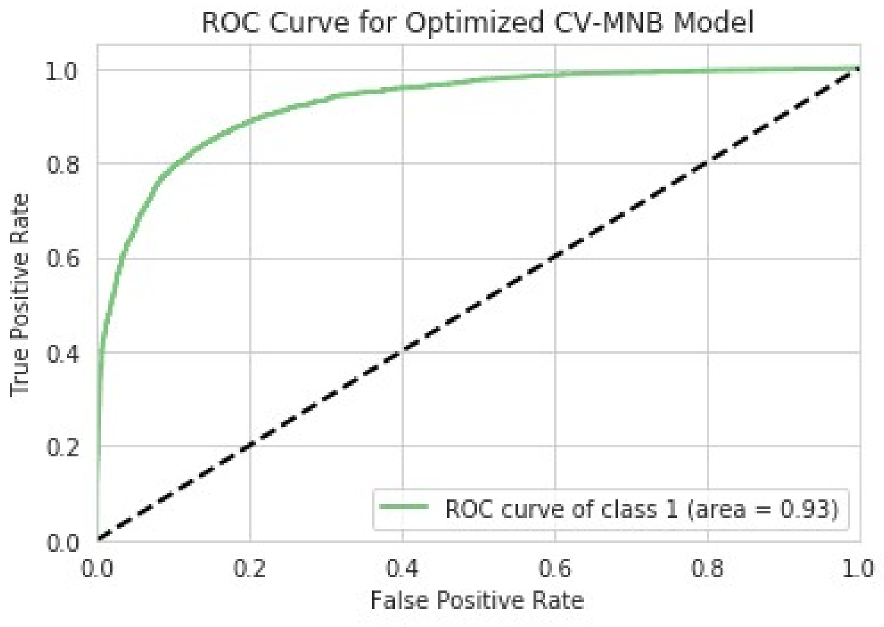
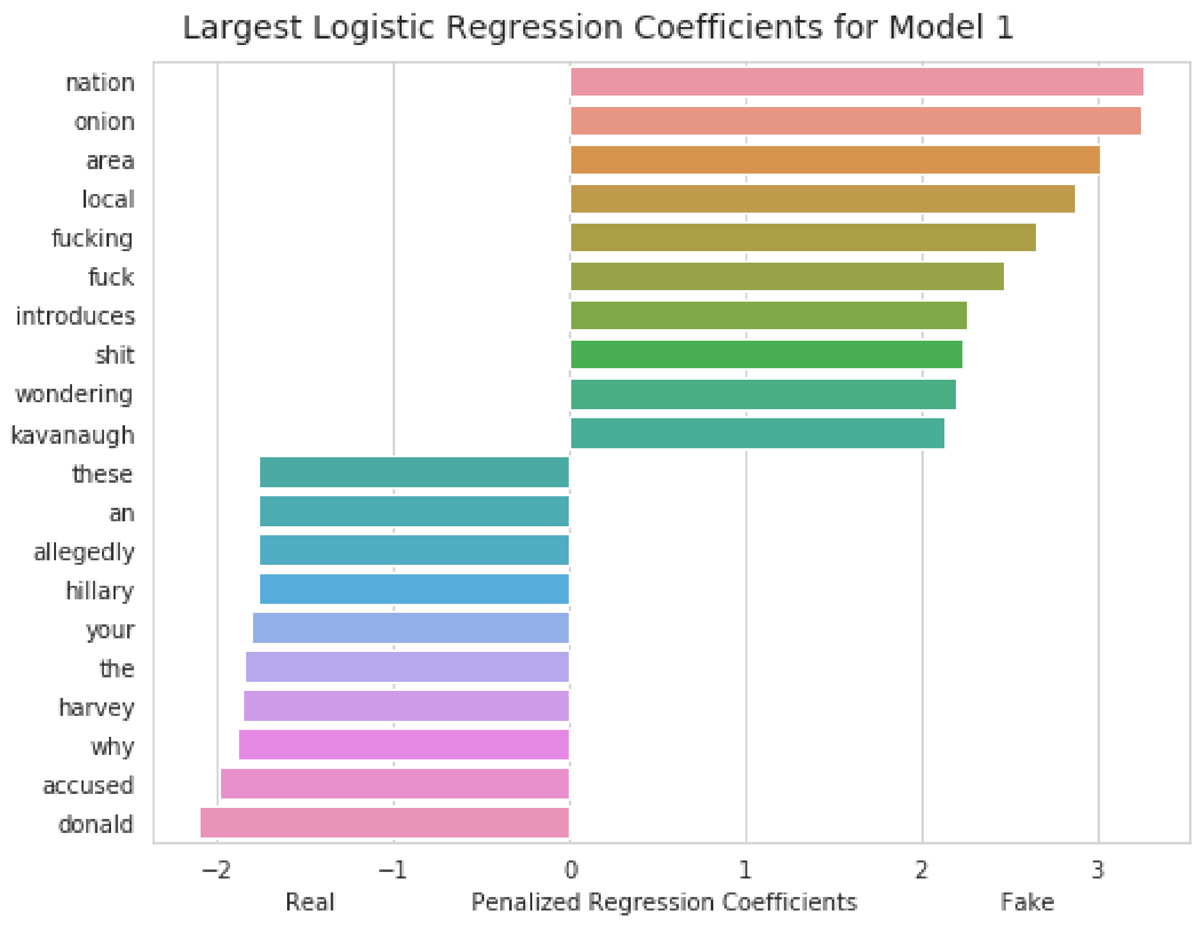


Figure 7: ROC curve for final model with the theoretical 'no-skill' model with AUC = .50 shown by a dotted line.

Another major consideration for a fake news production classifier is the need to periodically re-train the model as the word frequencies in news headlines are highly topical and will inevitably change quickly over time. The frequent appearance of names of current & recent US presidents showcases this problem well. In order to produce a reliable future proof fake news classifier a deeper understanding of the writing style may be more desirable despite the potential loss of accuracy and the increase in model complexity. Finally, as with many machine learning classifiers; the performance should be considered in context with that of humans. It is unlikely that a human would correctly identify 100% of news stories as real or fake based on headline alone and an accuracy of close to 85% is almost certainly high enough to prove useful.

Supplementary Material

The BoW Logistic Regression (Model 1) was used to investigate which words were particularly powereful in resolving real & fake news by analysing their regression co-efficients. This portion of the analysis did reveal that the coarse and direct language in fake news was a defining characteristic. The presence of more measured language e.g. ‘allegedly’, ‘accused’ etc in the list of words most associated with real news as also unsurprising.


Largest Logistic Regression Coefficients for Model 1
nation
onion
area
local
fucking
fuck
introduces
shit
wondering
kavanaugh
these
an
allegedly
hillary
your
the
harvey
why
accused
donald
Real
Penalized Regression Coefficients
Fake