Final Report

Thomas Cutts
tc3g20@soton.ac.uk

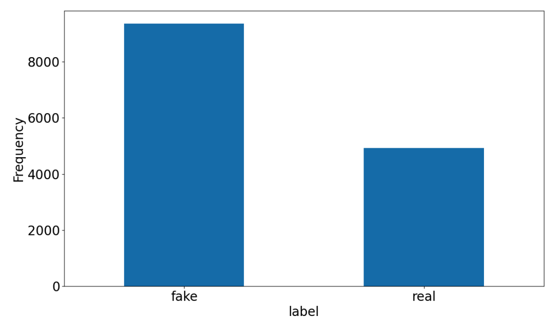
1 Introduction

1.1 Problem Characterisation

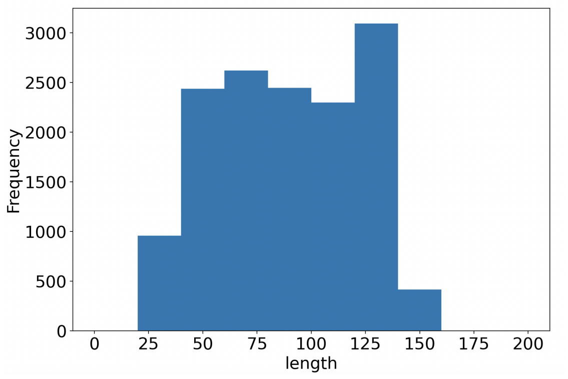
Many fake posts exist on Twitter, which can misinform the user-base. This project aims to classify tweets as real or fake based upon a corpus of tweet text and metadata. A post is considered fake if it contains multimedia that is: manipulated, synthetic or old and being presented as relating to a current event. Properties of the tweet text are assumed to impact the model (listed in Section 1.2). On the other hand, any ID numbers are assumed to have no impact on the model; as their purpose is for database management rather than holding any valuable information regarding the tweet. Usernames would matter to the model – but the lack of user features in the dataset (such as number of followers, previous tweets, etc.) prevent it from doing so. Also note that the model is trained on **text and metadata only** – not on images. The f1 score for the algorithm will be the official metric of success, however other evaluation techniques will be explored in Section 3. Since social media platforms such as Twitter are people's primary news sources nowadays, it is of the utmost importance to prevent misinformation. Especially since many credible news organisations use Twitter themselves to find information on breaking news [2]. Therefore this project aims to provide a verification tool for journalists to ensure the credibility of their sources.

1.2 Data Characterisation

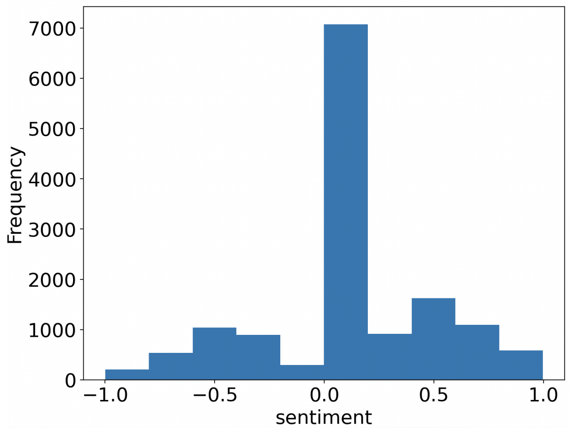
The dataset is structured, in tabular format. More specifically, it is a tab-delimited txt file. The dataset originally had three classification classes (fake, humor and real) but for the purpose of this task all humor posts will be considered as fake. The training set contains 14,483 labelled data points, therefore availability/volume is not an issue – in spite of the class imbalance. However, it is by no means considered 'big data', making deep neural networks less suited to this problem. The data has no missing values and very few duplicates, which adds to its quality. But the data is 8 years old, and although tweeting has not changed much since then, the quality of fake posts most likely have. So the classifier developed for this dataset will likely not perform as strongly when classifying more modern fake posts. The data is imbalanced, as shown in Figure 1(a). Fake posts have roughly double the representation compared to real posts, which is not representative of tweets in general. Furthermore, Figure 1(b) highlights the bias towards longer tweets, with the majority of tweets being around 130 characters. Figure 1(d) shows the clear bias towards tweets with a neutral sentiment.



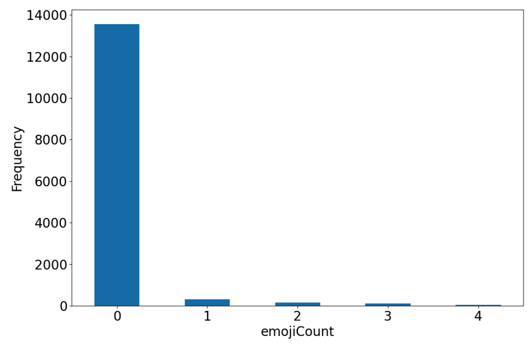
(a) Class distribution



(b) Length distribution



(c) Sentiment distribution



(d) Emoji count

Figure 1: Dataset characteristics

Features extracted from the tweet text include:

(1) Length
(2) Sentiment
(3) TF-IDF
(4) Emoji Count
(5) Exclamation mark count
(6) Hashtag count
(7) Question mark count
(8) Profanity count
(9) URL count
(10) Mentions count

These are the features used to train the model. They are all properties of the tweet text. All of these features have been assumed to be informative and non-redundant. This is under the presumption that there tends to a be a difference in style/content of fake tweets. A contrived example of this would be fake tweets containing more hashtags on average than real ones. I used the TfidfVectorizer [5] to generate the Term Frequency-Inverse Document Frequency matrix. It reflects the importance of words in the tweet text. The TF-IDF score increases as the number of occurrences of a word in a given tweet text increases, while also taking into account the overall frequency of that word in the entire corpus, to account for commonly used words. For determining the sentiment of a given tweet, I used VADER [4]. It is a sentiment analysis tool that is specialised for social media posts, making it a fitting choice for this project. It outputs a normalised number, where a negative number would represent an overall negative sentiment, a positive number representing a positive sentiment, and 0 representing a neutral sentiment. Features such as Hashtag and URL count were inspired by [1]. I used regular expressions to extract these features from the text. Additionally, the computational speed required for the project is feasible. This is due to the linear time complexity of training Naive Bayes (see Section 2.4) – $O(nkp)$ – where n is the number of data points, k is the number of classes, and p is the number of features.

2 Algorithm Design

2.1 Preprocessing

To calculate the TF-IDF matrix I used typical NLP preprocessing techniques:

(1) Converted text to lower case
(2) Removed punctuation
(3) Removed URLs
(4) Removed emojis
(5) Tokenised
(6) Removed stopwords
(7) Lemmatised

One extra precaution I took was removing apostrophes after removing stopwords (separately from other punctuation). This was to avoid stopwords like *don't* being converted to *dont* – allowing it to go unremoved. It is worth noting that I performed sentiment analysis before the preprocessing steps listed above. The reasoning for this is explained in [4]. The sentiment-analyser takes into account features such as capital letters, emojis and punctuation. So preprocessing its input would hinder its performance.

I originally planned on tackling the class imbalance by *resampling* the dataset. However, I did not use this technique in the end due to its negative impact on performance. Another preprocessing step that was omitted was translating the tweet text – as the dataset contains multiple languages. I tried libraries such as googletrans [3] – but it was too slow. I suspect it was because it was making HTTP requests to the google API, which for thousands of data points is a bottleneck. By not translating the data, it increased the dimensionality of the TF-IDF matrix; therefore worsening performance.

2.2 Feature selection

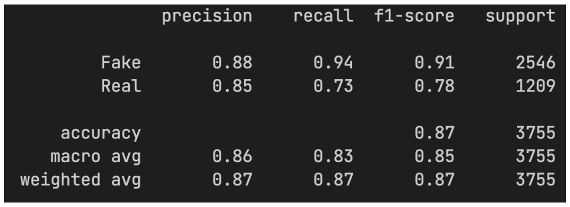
I used the SelectKBest class from sklearn [5] to perform feature selection. It is a filter based method, meaning that it uses a statistical measure to determine the relevance of features based on their correlation with the outcome. In my case it was the f-value. The result was that the two best features were the TF-IDF and positive sentiment analysis. This was likely due to the small amount of variance in the other features making them not very meaningful. An example of this is shown in Figure 1(d).

2.3 Dimensionality reduction

The great thing about Naive Bayes is that it handles high-dimensional data easily and does not require explicit dimensionality reduction. Note that the preprocessing and feature selection steps listed previously reduce the dimensionality of the data by proxy. I did attempt to perform Non-Negative Matrix Factorisation (NMF), which produces only non-negative values (unlike PCA) as Multinomial Naive Bayes cannot process negative inputs. However it worsened performance as discussed in Section 3.

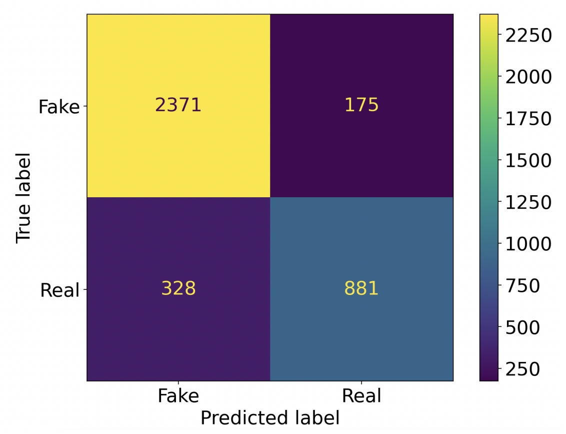
2.4 Algorithm choice

I chose a Naive Bayes classifier for several reasons. Firstly, because they are suited to text classification problems. They are also computationally efficient and don't require much hyperparameter tuning – as they only have one hyperparameter. Seeing as I trained the model on my laptop, I did not have significant computational resources available. They are also very good at handling high dimensional data - which is important in my case as the TF-IDF matrix is very large (it has roughly 28000 features). It is also relatively insensitive to irrelevant features. The reasoning behind this efficiency is due the naive assumption that the features are conditionally independent which is not the case for our data set. It is obvious that words are conditionally dependent on each other in a sentence. For example, the probability of the word *tasty* occuring in a sentence is increased by the occurence of the word *food*. This naive assumption by the model can hinder performance. I chose the Multinomial Naive Bayes classifier because the scikit-learn documentation stated that it is suited to working with discrete data, which is the case for our data set.

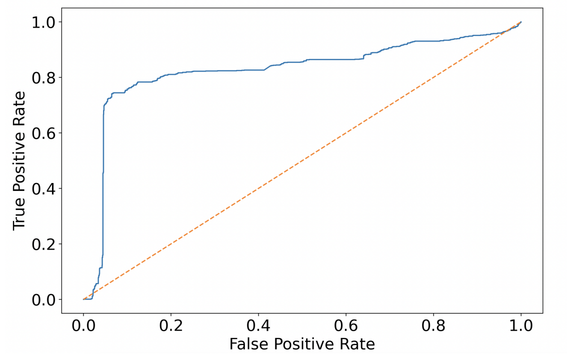


```
              precision    recall  f1-score   support

        Fake       0.88      0.94      0.91      2546
        Real       0.85      0.73      0.78      1209

    accuracy                           0.87      3755
   macro avg       0.86      0.83      0.85      3755
weighted avg       0.87      0.87      0.87      3755
```

(a) f1 score

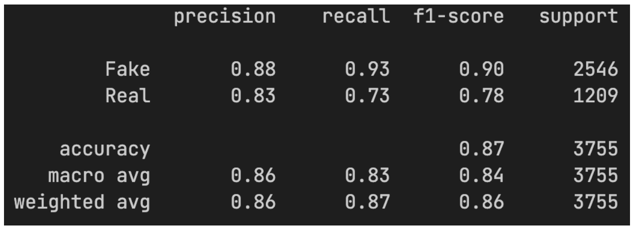


(b) Confusion Matrix



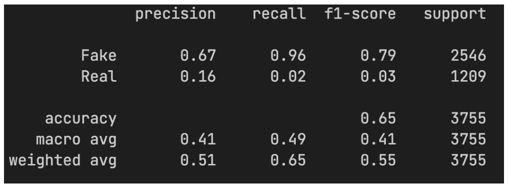
(c) ROC curve

Figure 2: Final model metrics



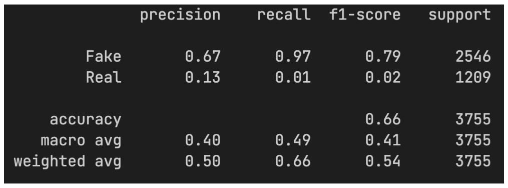
```
              precision    recall  f1-score   support

        Fake       0.88      0.93      0.90      2546
        Real       0.83      0.73      0.78      1209

    accuracy                           0.87      3755
   macro avg       0.86      0.83      0.84      3755
weighted avg       0.86      0.87      0.86      3755
```

Figure 3: TF-IDF only



```
              precision    recall  f1-score   support

        Fake       0.67      0.96      0.79      2546
        Real       0.16      0.02      0.03      1209

    accuracy                           0.65      3755
   macro avg       0.41      0.49      0.41      3755
weighted avg       0.51      0.65      0.55      3755
```

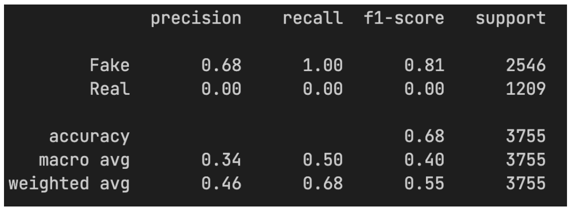
(a) Random Forest



```
              precision    recall  f1-score   support

        Fake       0.67      0.97      0.79      2546
        Real       0.13      0.01      0.02      1209

    accuracy                           0.66      3755
   macro avg       0.40      0.49      0.41      3755
weighted avg       0.50      0.66      0.54      3755
```

(b) XGBoost

Figure 4: Other models tried



```
              precision    recall  f1-score   support

        Fake       0.68      1.00      0.81      2546
        Real       0.00      0.00      0.00      1209

    accuracy                           0.68      3755
   macro avg       0.34      0.50      0.40      3755
weighted avg       0.46      0.68      0.55      3755
```

Figure 5: With NMF

3 Evaluation

The break-down of the model's f1 score is shown in Figure 2(a). The model achieves an average f1 score of 0.85. This is a strong score. It is clear that the model performs better at classifying fake tweets than real ones – shown by the difference in f1 score for both respective classes (0.91 vs 0.78). This is due to the skew in the data set towards fake tweets. The confusion matrix confirms this by showing how the model predicts fake roughly 2.5 times more often than real. The ROC curve shown in Figure 2(c) also reinforces this idea, as the area under the curve (AUC) is relatively small for 'real' predictions which suggests underfitting. Figure 3 shows the improvement with the addition of the sentiment-analysis feature. Although the improvement is marginal (f1 score increased by 0.01). Before settling on Multinomial Naive Bayes, Figures 4(a) and (b) show the performance of Random Forest and XGBoost. It is clear that they both performed poorly with TF-IDF. This is because both algorithms are based on decision trees which can be affected by the high dimensionality and sparsity of the feature space, inhibiting them from effectively splitting the data based on these features. Figure 5 shows how the application of NMF to reduce the number of components down to 10 severely impacted the score.

The *t-test* was used to determine whether the difference between the model and a random classifier is statistically significant. I used the scipy package [6]. With a p-value of 0.0 and a t-statistic of 60.5, it shows that it is *very* unlikely that the model's better performance is due to chance. Therefore the results *are* statistically significant.

4 Conclusion

In conclusion, out of the three models tested, Multinomial Naive Bayes saw the greatest performance. The biggest performance-booster however, was the use of TF-IDF. Future work includes implementing efficient translation of the text in the data set. This would not only standardise the data but also reduce the dimensionality, as there would be fewer unique words in the document. Additional next steps include developing the classifier further to process the image data. By limiting the model to processing only text, performance is very much hindered. The end goal would be to combine an image classifier, so both the tweet image and text are considered by the model. Furthermore, experimentation with different algorithms would be a future objective. It would be interesting to compare the results of a Support Vector Machine or a Deep Neural Network.

References

[1] Christina Boididou et al. “Challenges of Computational Verification in Social Multimedia”. In: *Proceedings of the 23rd International Conference on World Wide Web*. Association for Computing Machinery, 2014. ISBN: 9781450327459. DOI: 10.1145/2567948.2579323. URL: https://doi.org/10.1145/2567948.2579323.

[2] Christina Boididou et al. “Verifying Multimedia Use at MediaEval 2015 In MediaEval Benchmarking Initiative for Multimedia Evaluation”. In: Sept. 2015.

[3] Google. *googletrans 3.0.0*. 2020. URL: https://pypi.org/project/googletrans/.

[4] Gilbert E.E. Hutto C.J. *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. 2014. URL: https://github.com/cjhutto/vaderSentiment.

[5] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* (2020). DOI: 10.1038/s41592-019-0686-2.