COMP3222 Coursework Report
Peter Johnstone - 30184142
phj1g18@soton.ac.uk

1 Introduction and Data Analysis

1.1 Introduction

Fake and misleading content is becoming increasingly more present on social media, particularly in the aftermath of significant events. When reporting on significant events, journalists need to filter large volumes of posts to find real and important information such as eyewitness content. Therefore it is important there are tools that can automatically judge if social media posts are truthful in order to aid this process.

The goal of this project is to develop a machine learning algorithm that can classify social media posts from twitter as either ’real’ or ’fake’ from the text content of the tweet. In this context, a tweet is labelled as ’real’ if the media included in the post legitimately represents the event it refers to, and is labelled as ’fake’ if it does not. This is because the dataset used is from the MediaEval 2015 "verifying multimedia use" challenge, which included both real and misleading media with the posts. However, for this assignment, the media have been stripped from the dataset to limit the scope of the project to just the tweet contents.

1.2 Data Analysis

1.2.1 Data Format

Training and testing datasets are provided in the form of a .txt file, where columns are delimited by tabs and rows are delimited by new lines. The attributes of the datasets are outlined in table 1. The only difference between the training and testing datasets is that the training dataset entries can either have a ’real’, ’fake’, or ’humor’ label, whereas the test set entries can only be ’real’ or ’fake’. For this project, ’humor’ labelled posts are to be considered as ’fake’.

Attribute	Type
tweetId	Integer
tweetText	String
userId	Integer
imageId(s)	String
username	String
timestamp	String
label	String

1.2.2 Data Volume

The training dataset file has a size of 2.53 MB and contains 14483 tweets. The testing dataset has a size of 759KB and contains 3781 tweets. This is a relatively small dataset, which could limit the possible approaches for this project, but also means it will be relatively fast to run iterations of an algorithm for this dataset.

1.2.3 Data Quality

The training dataset is of relatively high quality in the sense that there are no missing values in the entries, which could otherwise cause issues. However, the presence of encoding characters, emojis, and spelling errors in the tweet text of some entries could detract from the quality of the dataset and will require pre-processing for the algorithm. Additionally, there are 2843 duplicated tweets which could skew classifier performance in some way.

1.2.4 Data Bias

This part of the data characterisation stage is focused on the training data only: the test data was set aside to avoid a snooping bias.
As shown in figure 1, 'fake' labelled posts are the most common in the dataset. If 'humor' posts are considered to be 'fake' as a classifier would treat them, then the dataset is biased towards 'fake' posts, which could result in an algorithm overfitting towards 'fake' labels, or for the 'fake' F1 scores to be much higher than the F1 score for 'real' posts. To deal with this bias, some 'fake' entries could be removed from the training dataset to create a more even distribution.

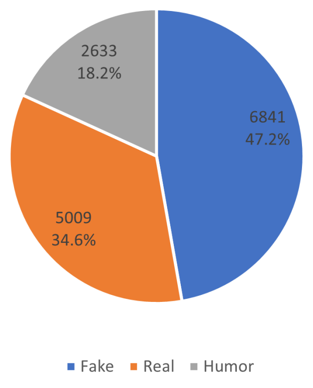


Figure 1: Label distribution in training dataset

There is a heavy bias with regards to the events discussed in the tweets of the training dataset. Hurricane Sandy is the most common topic by a significant margin, accounting for 70.6% of all unique posts. Among posts with a 'real' label, the bias is even clearer. This could lead to overfitting towards tweets concerning Hurricane Sandy or even tweets about hurricanes or

New York City. This bias could be mitigated using techniques such as Parts-Of-Speech (POS) tagging, which adds labels to phrases relating to the grammatical category they fit into or by oversampling some of the less frequently described events. Despite there being a wide variety

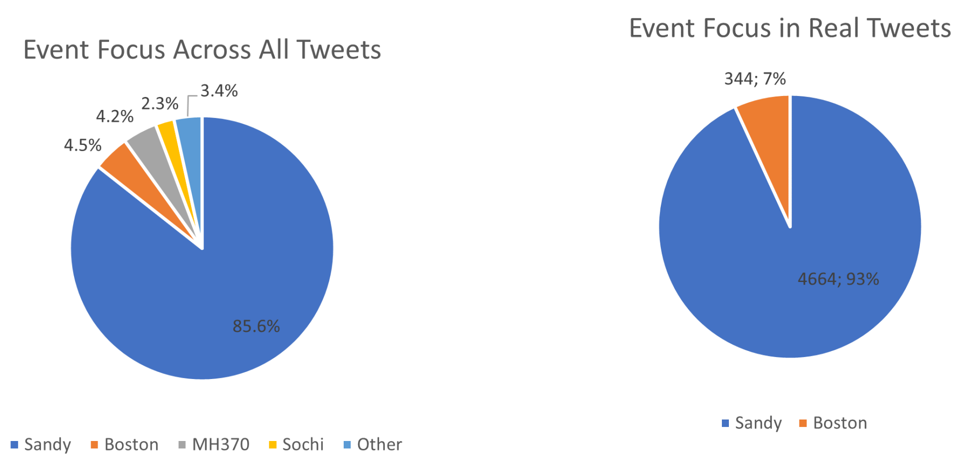


Figure 2: Pie charts comparing the tweet topic distribution across all entries and real entries

of languages used in the tweets of the training dataset, it is skewed towards English, as shown in figure 3. Interestingly, Tagalog is the 3rd most common language in the training dataset despite only being the 58th most commonly spoken language [1]. Because there is more data to work with, it is likely that the classifier will perform better on languages that have more representation in the training dataset. However, this does not mean that each language should have equal representation within the dataset since that would result in the vast majority of entries being discarded, severely limiting the performance of the classifier. One approach to

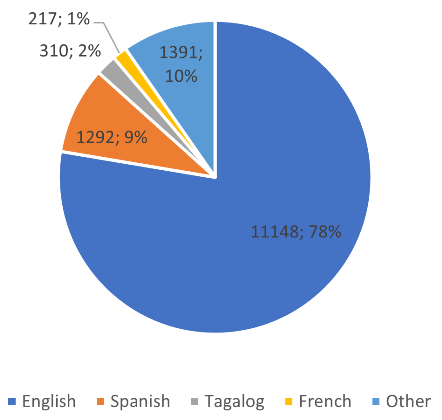
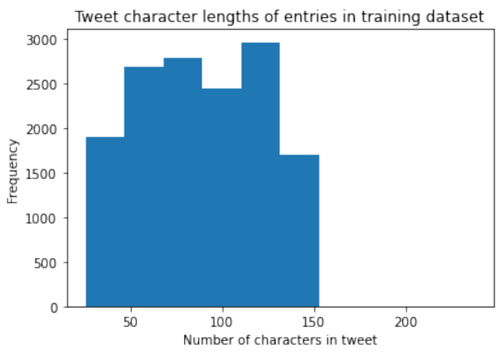


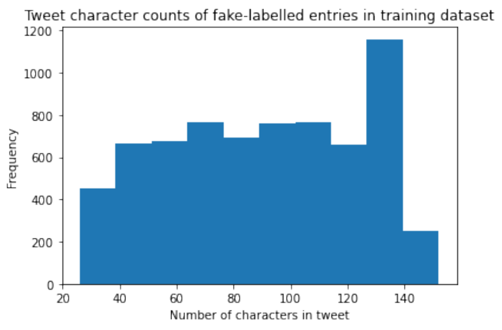
Figure 3: Language distribution in training dataset

make the classifier perform better on languages with less representation is to translate every tweet into English in the pre-processing stage, although this is a time-consuming procedure.

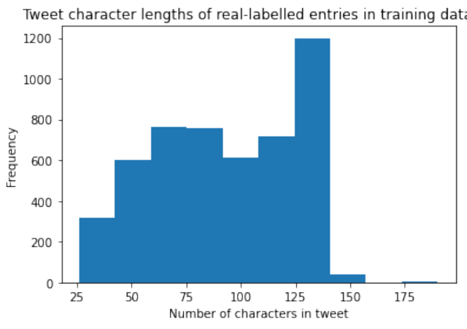
As figure 4a shows, the tweets in the training dataset have a relatively normal distribution with regards to character count, but figures 4b, 4c, and 4d show that there is a slight disparity between real, fake and humor partitions. Among real and fake tweets, the distribution peaks around the 140 character limit Twitter imposed at the time. In contrast, 'humor' labelled tweets tend to be much shorter. This suggests that it may be wise to treat the 'fake' and 'humor' classes differently in the training phase.



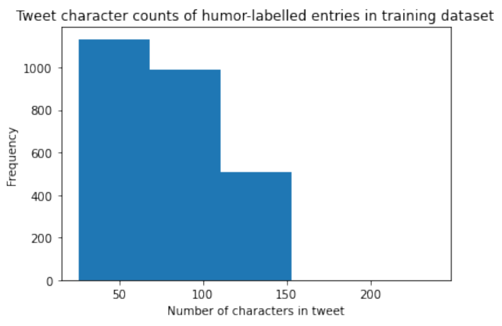
(a) All entries



(b) Fake entries



(c) Real entries



(d) Humor entries

Figure 4: The distribution of tweet character counts in different partitions

It would have been useful to characterise the users in the datasets, in order to find attributes such as how many followers they have, or if they are a verified user. However, since this information is not available as we cannot visit user profiles, so we cannot do this.

2 Algorithm design

2.1 Hypothesis Formulation

After analysing the training dataset, it was clear that classes of posts tended to use a similar vocabulary and speech patterns. Therefore, a good strategy for classifying tweets would be to learn the natural language clusters of each class, and label tweets in the test data according to the class they fit best. A bag of features approach works well for this strategy. A neural network approach would also be suitable for this problem, but was not chosen as it appeared more difficult to implement, especially when considering I was already familiar with the bag of features model, having used it in another project. A deep learning approach was deemed unsuitable for this problem due to the small size of the dataset being a limiting factor.

2.2 Pre-processing

2.2.1 Data cleaning

All tweets in the training and testing data go through a stage of cleaning before they are used by the classifier. A regular expression was used to remove any URLs linking to the media of a tweet, since they do not contribute anything useful to a classifier, as well as special characters such as hashtags and mathematical symbols, since they both do not contribute useful information and can lead to the same words having multiple entries. Tweets are then converted to lower case using an inbuilt Python string function, as otherwise the same word with a different capitalisation can be added twice. These steps help remove irrelevant information and keep the dataset sparse. However, I opted to keep the @ character, since it relates to Twitter usernames which could be useful, and emojis, since they appear frequently and convey meaning.

2.3 Transforming

In the final algorithm design, all 'humor' labels are replaced with a 'fake' label. Despite the differing characteristics between the two classes, the classifier performed better when given binary classes. Although other transformation techniques were considered, they were ultimately not used. It was decided that tweets in other languages should not be translated into English. Although this would address the language bias, as by making everything the same language you do not have to consider language distribution, translating every tweet in the training and test datasets would make the algorithm's computation speed much slower, and there is also a small risk of incorrect translation. As a consequence, POS tagging, stemming (converting declensions of the same word to a common root) and stopword removal (filtering out common words such as 'a' and 'the') were not used to transform tweets, since their implementation in multiple languages would be too difficult. It should be acknowledged that these steps would have likely improved classifier performance, and could be implemented as a future improvement.

2.3.1 Tokenising and n-grams

Once cleaned, tweets are tokenised and split into a list of n-grams, which are a sequence of n tokens. Tokenising allows a machine learning algorithm to learn natural language clusters at the word level when combined with techniques such as TF or TF-IDF, while n-grams can provide useful information when key phrases occur often, as opposed to just individual words. This step was done using Scikit-learn's CountVectorizer, and it was found that using a range of unigrams to 7-grams in conjunction with a limit of 17500 features lead to the best classifier performance.

2.4 Feature Selection

The'tweetId', 'userId', 'imageIds(s)', 'username' and 'timestamp' columns in the dataset were all ignored since they do not provide useful information for classification. Only the 'tweetText' and 'label' columns of the dataset are used. After a tweet goes through pre-processing, the number of occurrences of each word and n-gram in the tweet is calculated using the CountVectorizer. This allows a numerical feature vector to be generated from the text data. A more useful feature vector in Term Frequency-Inverse Document Frequency (TF-IDF) can be generated from this. TF-IDF identifies words and n-grams in a tweet which appear more often in comparison to the rest of the tweets, which is a better determiner of importance than simply how many times a word appears in a tweet. TF-IDF used with n-grams has achieved high scores in text classification problems, so it was chosen as the main feature in this algorithm [2].

2.5 Dimensionality Reduction

Given that the only feature space is relatively small, dimensionality reduction did not seem appropriate. If the feature space was greater than 100,000, then it would have been considered.

2.6 Machine Learning Algorithm

A multinomial naïve Bayes classifier for this problem for several reasons. It is known to work well on text classification problems; in particular it works well when combined with TF-IDF [3]. It is easy to implement, with only 3 classifier parameters in its Scikit-learn implementation - *alpha*, *fit_prior* and *class_prior* - that can be left as their default values and still expect high performance. On the other hand, Linear Support Vector Machines, while known to perform well on text classification problems, will suffer from a poor recall score unless the right set of parameters is chosen [4]. While they can still perform well in text classification, non-linear classifiers such as Decision Trees and K Nearest Neighbours tend to perform slightly worse than linear classifiers, so they were not considered for the initial algorithm designs [2].

3 Evaluation

3.1 Evaluation metrics

As specified in the coursework assignment description and FAQ, the micro F1 score is the most important evaluation metric for this problem. To calculate the micro F1 score, I used *metrics.f1_score()* from Scikit-learn, providing the ground truth labels in the test dataset, the classifier's predictions, and an argument telling the function to return the micro F1 score. In order for a classification algorithm to receive a high F1 score, it must achieve both a high precision and high recall.

3.2 Evaluation of algorithm design iterations

The first algorithm design evaluated involved an *SGDClassifier* with a hinge loss function, l2 penalty and 0.001 alpha value. It made use of TF-IDF of unigram features, with the only pre-processing step being the conversion of tweets to lower case. After the predictions were made, all 'humor' labels were changed to 'fake'. This achieved a micro F1 score of 0.613. While the recall of 'fake' labels was high at 0.86, the recall of 'real' labels was only 0.13.
The next iteration swapped the *SGDClassifier* for a *MultinomialNB* classifier with a *alpha* value of *1*, a *fit_prior* value of *True* and a *class_prior* value of *None*, but kept the other aspects as they were. This achieved a micro F1 score of 0.793, a significant improvement largely down to the increased recall of 'real' elements.
The final iteration made changes at the pre-processing stage, while keeping the classifier the same. URLs and special characters were removed using a regular expression, and changed the 'humor' labels to 'fake' in the training set instead of in the predictions. These changes increased the micro f1 score to 0.83. This was the design I settled on, but I wanted to find the best parameters to maximise the final F1 score.

3.3 Evaluation of different algorithm configurations

The two parameters that were configured were the range of n-grams that tweets were to be broken up into and the maximum number of features in the matrix of tokens. This was because it became clear during testing that each set of n-grams had an optimal maximum features value, and so I wanted to find the pairing of the two parameters which would result in the highest F1 score. To do this, I built a classifier pipeline, where the *ngrams* and *max_features* arguments to construct the *CountVectorizer* were adjusted through iteration. At each configuration, the micro F1 score was calculated and tabulated in table 1.
The intervals for the maximum feature argument were chosen so that a rough estimate of the best value could be found. The n-gram ranges were chosen to see how the F1 score changes as the maximum n-gram increases - as the table shows, increasing the minimum n-gram length decreases the F1 score because it performs worse on 'real' tweets, possibly a consequence of label bias as there may not be enough data to work with.

A second set of iterations was performed to find the highest F1 score for each n-gram range. As table 2 shows, 0.861 was the highest F1 score recorded for the final classifier, achieved with a (1,7) n-gram range and a maximum of 17500 features in the dictionary. Given that the highest scores for this coursework were over 0.9, there is still some room for improvement, although it is still an acceptable score. It should be noted that this classifier takes noticeably longer to run in comparison to configurations with a smaller n-gram range.

N Grams	**Max Features**								
	2000	3000	5000	7500	10000	15000	20000	30000	40000
(1,1)	0.616	0.816	0.681	0.808	0.805	**0.833**	0.833	0.833	0.833
(1,2)	0.631	0.649	0.811	0.820	0.801	**0.822**	0.770	0.761	0.670
(1,3)	0.590	0.652	0.626	0.745	**0.831**	0.827	0.765	0.774	0.771
(1,4)	0.592	0.652	0.633	0.628	**0.838**	0.836	0.769	0.766	0.788
(1,5)	0.597	0.655	0.652	0.619	0.755	**0.837**	0.799	0.781	0.787
(1,6)	0.599	0.655	0.646	0.655	0.650	**0.853**	0.799	0.670	0.679
(1,7)	0.611	0.655	0.646	0.656	0.649	**0.853**	0.793	0.778	0.681
(1,8)	0.612	0.596	0.645	0.655	0.637	**0.853**	0.789	0.781	0.785
(2,2)	**0.662**	0.608	0.606	0.565	0.565	0.566	0.567	0.575	0.603

Table 1: F1 scores of different configuations

N Grams	**Optimal Max Features**	**Highest F1 Score**
(1,1)	15000	0.833
(1,2)	17000	0.833
(1,3)	14500	0.846
(1,4)	10500	0.843
(1,5)	16500	0.855
(1,6)	15500	0.855
(1,7)	17500	0.861
(1,8)	16000	0.856

Table 2: Optimal max features value for different n-gram ranges and their F1 score

If a fast computation speed is highly desired, accuracy can be sacrificed for it by choosing one of the other configurations.

4 Conclusion

In this report, I have summarised the approach taken to build a classifier for the MediaEval dataset. A problem definition was described and a detailed analysis on the dataset was performed. The algorithm design has been fully described and justified. The stages of iterative development of the algorithm have been described, and evaluations in the form of F1 scores have been performed at each stage of development and at different configurations of the final design. One lesson learnt from this project is the importance of data cleaning and pre-processing, which have as big of an impact on classifier performance as the classifier parameters. While the F1 scores for the final design are acceptable, they can still be improved upon. As future work, extra steps can be taken at the pre-processing stage by translating every non-English tweet into English and then performing stemming and POS tagging.

References

[1] David M Eberhard, Gary F Simons, and Charles D Fennig. Ethnologue: Languages of the world., Dec 2020.

[2] Hadeer Ahmed, Issa Traore, and Sherif Saad. Detection of online fake news using n-gram analysis and machine learning techniques. In Issa Traore, Isaac Woungang, and Ahmed Awad, editors, *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, pages 127–138, Cham, 2017. Springer International Publishing.

[3] Ashraf M. Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In Geoffrey I. Webb and Xinghuo Yu, editors, *AI 2004: Advances in Artificial Intelligence*, pages 488–499, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] Emmanouil Ikonomakis, Sotiris Kotsiantis, and V. Tampakas. Text classification using machine learning techniques. *WSEAS transactions on computers*, 4:966–974, 08 2005.