COMP3222: Machine Learning Technologies Final Report

Lukas Kakogiannos

January 2024

1 Introduction

1.1 Problem Description

In the digital age, the rapid dissemination of information through social media platforms has become a double-edged sword. While it enables the swift sharing of news and updates, it also creates a fertile ground for the spread of misinformation and manipulated content. One particularly concerning phenomenon is the propagation of fake images or the presentation of real images in a false context through viral social media content. This issue becomes especially pronounced in the aftermath of high-impact events, where emotions run high, and the thirst for information is insatiable.

As the immediacy of social media encourages users to share content without thorough fact-checking, a cascade effect often occurs, amplifying misleading information within minutes. This challenge necessitates a careful and systematic investigation to debunk false narratives and discern whether shared multimedia accurately represents real information. This process is crucial not only for preserving the integrity of public discourse but also for preventing the potential consequences of baseless claims or manipulated visuals.

With the lack of publicly accessible tools, the goal is to tackle the problem by designing machine-learning pipelines with different algorithms that can classify social media posts as 'real' or 'fake'. The aim is to assist news professionals, such as journalists, in verifying their sources and ensuring the accuracy and objectivity of their reporting. To train the algorithms, the MediaEval 2015 "verifying multimedia use" dataset will be used, as it contains training and test sets that come in a text format (the values are tab-separated).

1.2 Data Analysis

The first step is to analyse the training set, it comprises 7 columns that include:

- tweetId
- tweetText

- usedId
- imageId(s)
- username
- timestamp
- label

the set has a total of 14483 entries, of which none of them contain null values.

This is a classification problem, as the algorithms' goal is to identify which of the two classes the social media posts fall under, however, the dataset maps the data to three different labels, 'fake', 'real', and 'humour'. The next step is to get rid of the 'humour' label, the most sensible alternative is to change all the data that is under the 'humour' label into the 'fake' label, turning the problem into a binary classification one.

The figure below describes the new distribution of tweets by label:

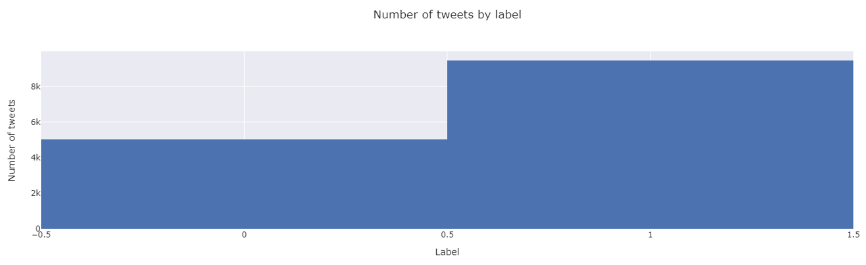


Figure 1: the training set contains 5009 'fake' tweets, and 9474 'real' ones

Another option would be to analyse the length of tweets since this factor can easily be used as a feature for the algorithms' learning process. By using the 'tweetText' column we can easily find out the length of all tweets and place it under a new column called 'length', as denoted below:

```python
# Calculate tweet lengths
trn_data['length'] = trn_data['tweetText'].str.len()
```

Listing 1: Code used to find length of all tweets in the training set

The figure below describes the distribution of tweets by their length:

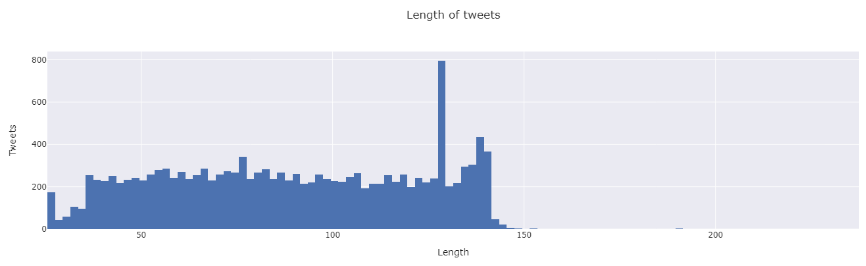


Figure 2: it is noticeable that most tweets contain below 142 characters

2 Pipeline Design

The design of the pipeline is separated into three different steps, pre-processing, feature selection, and deciding on the classification algorithms.

2.1 Pre-Processing

Pre-processing is a crucial initial step in data preparation for the pipeline. It involves cleaning and transforming raw data into a format suitable for training. This includes handling missing values, removing irrelevant information, standardizing data formats, and addressing inconsistencies. Proper pre-processing enhances the model's performance, improves interpretability, and mitigates potential issues stemming from irregularities or noise in the raw data.

When it comes to the training set, since there are no null values, the first step was to address inconsistencies in the data by changing everything to lowercase and removing any URLs and non-alphabetical characters. Additionally, the following piece of code was used to remove any duplicate entries from the training set.

```
# Removing duplicate rows
trn_data = trn_data.drop_duplicates(subset=['clnTweet'], keep='first').
    reset_index(drop=True)
```

Listing 2: Code used to filter duplicates

Moreover, removing emojis (method found online, cited in the code) and stop words were the two next steps while preparing the data, stop words are common words that are often filtered out during the pre-processing of natural language text data, examples of stop words include words like 'and', 'the', 'is', and 'in'. This was done to improve feature extraction and handle sparsity, this will help in future steps when using techniques such as Bag of Words or a term frequency-inverse document frequency transformer.

The new length of tweets was stored in a new column named 'cleanlength', the figure below shows the distribution of tweets by their new length:

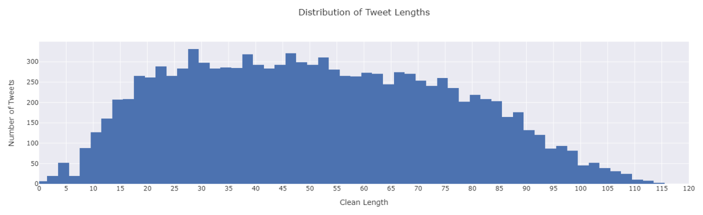


Figure 3: the new graph appears to be much closer to a normal distribution

The final step of pre-processing was removing all non-english tweets, this choice was made for three specific reasons:

1. Translation of text data requires additional resources, such as translation models or APIs. Having limited resources, removing non-English entries was a more feasible option.

2. Machine translation is not perfect and can introduce errors or inaccuracies. By removing non-English text, potential distortions in the meaning are avoided and the integrity of the original data is maintained.

3. Some languages have unique cultural and contextual nuances that may not be accurately captured through translation. For applications where preserving the original context and cultural nuances is crucial, removing non-English entries ensures that these subtleties are not lost.

To facilitate language detection, tweets that contain less than 4 characters were removed from the training set. However, to make sure that the training set would still have enough entries after the language filtering, a visualisation of the training set was created through a graph and a pie chart, as shown below:

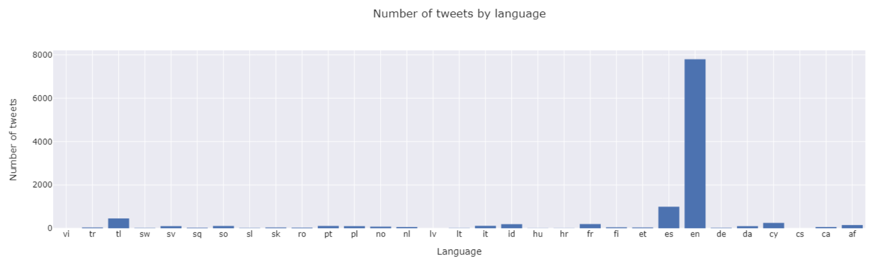


Figure 4: the graph shows all the languages included in the training set

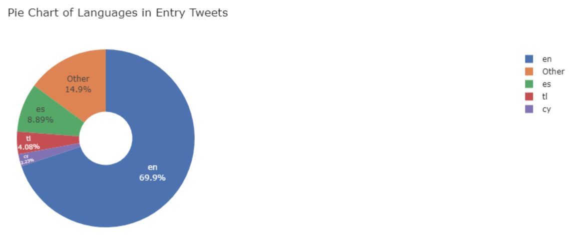


Figure 5: the pie chart displays the 4 most common languages in the training set and their percentage

2.2 Feature Selection

When selecting possible features to be used, a possible idea is to separately analyse the length of 'real' and 'fake' tweets, as the length of the text could be indicative of the writing style, with fake tweets potentially being overly verbose or overly concise, depending on the nature of the misinformation. Furthermore, the length of the tweet may reflect the complexity of the narrative. Fake tweets might attempt to simplify or overly complicate the message to manipulate the reader. The two figures below show the length of 'fake' and 'real' tweets, respectively:

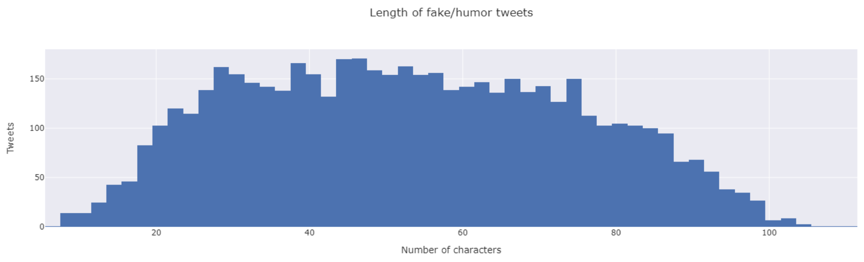


Figure 6: graph displaying the distribution of fake tweets

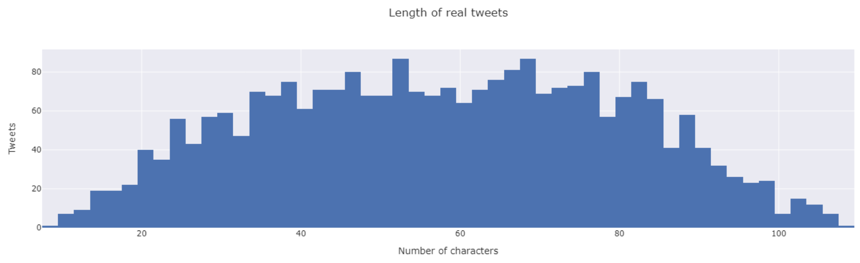


Figure 7: graph displaying the distribution of real tweets

Since there is a minimal difference between the distributions shown above, with 'real' tweets containing a slightly larger number of average characters per tweet, using this as a possible feature is not specific enough. However, while focusing on textual content, another possibility is to focus on the frequency of specific words, as the frequency of specific words or phrases can reveal linguistic patterns that are characteristic of either real or fake content. For instance, certain deceptive or misleading language may be more prevalent in fake posts.

To successfully analyse the frequency of words, the following code was used:

```
CountVectorizer(), # Convert text to a matrix of token counts
TfidfTransformer(), # Transform a count matrix to a normalized term-frequency
    or term-frequency times inverse document-frequency representation
```

Listing 3: Code that finds and weighs words based on their frequency

The objective in opting for TF-IDF over the raw frequencies of a token's occurrence in a specific document is to diminish the influence of tokens that frequently appear throughout a given corpus. These tokens are deemed less informative compared to other tokens that are present in a smaller fraction of the training corpus [scikit-learn developers, 2024].

2.3 Machine Learning Algorithms

When deciding which algorithms to train on the processed data set, four possible choices were considered, Multinomial Naive Bayes (MNB) [McCallum and Nigam, 1998, Bird et al., 2009], Random Forest [Islam et al., 2019, Pranckevivcius and Marcinkevivcius, 2017], Stochastic Gradient Descent (SGD) [Géron, 2017], and Support Vector Machine (SVM) [Géron, 2017]. The two chosen algorithms were MNB and SGD, explained in detail in sections 2.3.1 and 2.3.2, respectively.

2.3.1 Multinomial Naive Bayes

1. MNB is particularly effective for text classification tasks, especially when dealing with sparse data such as bag-of-words representations since it

assumes independence between features[Watson, 2001], making it well-suited for high-dimensional data.

2. MNB is computationally efficient and has fast training times. therefore, it is well-suited for large datasets and can handle text classification tasks with a high number of features.

2.3.2 Stochastic Gradient Descent

1. SGD is suitable for large-scale and sparse datasets. It's efficient for text classification tasks and can handle high-dimensional feature spaces. [Géron, 2017]
2. Depending on the specific model used, SGD can provide interpretable coefficients, especially with features like TF-IDF in text classification.
3. SGD can be more robust to noisy data and outliers when compared to other algorithms such as Random Forest. The iterative nature of SGD allows it to adapt to changing data distributions.

Finally, both algorithms typically have lower memory requirements compared to Random Forest, making them more suitable for this environment.

3 Evaluation

Evaluation is split into two different sections, confusion matrices, and F1 scores.

3.1 Confusion Matrices

Confusion matrices provide a detailed breakdown of a model's performance by displaying how many instances were correctly or incorrectly classified into different classes [Zhou, 2021]. This includes True Positives (correctly predicted positives), True Negatives (correctly predicted negatives), False Positives (incorrectly predicted positives), and False Negatives (incorrectly predicted negatives). The figures below display the confusion matrices for both MNB and SGD, respectively:

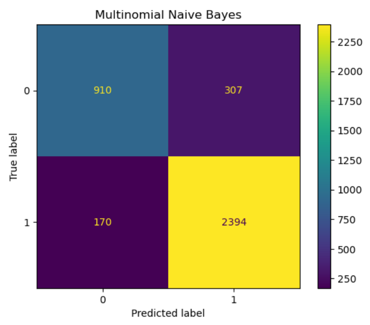


Figure 8: Confusion Matrix for MNB

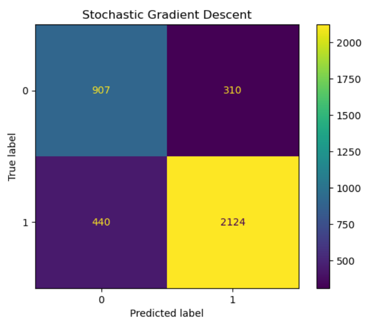


Figure 9: Confusion Matrix for SGD

From the matrices above, it can be observed that the classification was mostly successful, as the number of true positives (TP) and true negatives (TN), is way higher than the false positives (FP) and false negatives (FN).

Furthermore, using these four values, various evaluation metrics can be derived [Zhou, 2021], including:

- Accuracy - The proportion of correctly classified instances out of the total.
- Precision - The ability of the model to correctly identify positive instances among all instances it predicted as positive.
- Recall - The ability of the model to capture all positive instances in the dataset.
- F1 Score - The harmonic mean of precision and recall, providing a balance between the two.

The metrics that will be needed for the rest of the evaluation are Precision, Recall, and F1 score. These are calculated with the equations below:

$$Precision = TP/(TP + FP) \tag{1}$$

$$Recall = TP/(TP + FN) \tag{2}$$

and, finally:

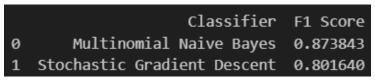
$$F1Score = 2 * (Precision * Recall)/(Precision + Recall) \tag{3}$$

3.2 F1 Scores

The F1 score is a metric that combines precision and recall into a harmonic mean. It provides a balanced assessment of a model's performance, especially when there is an imbalance between classes or when both false positives and false negatives need to be minimised. Scikit-learn contains a function that calculates the F1 score as follows:

```
f1_multi = f1_score(tst_data['label'], multinomial_pred, average="micro")
f1_stochastic = f1_score(tst_data['label'], stochastic_pred, average="micro")
```

Listing 4: Code used to calculate F1 score of both predictive models on the testing set



```
                   Classifier  F1 Score
0     Multinomial Naive Bayes  0.873843
1  Stochastic Gradient Descent  0.801640
```

Figure 10: Table displaying F1 scores for MNB and SGD

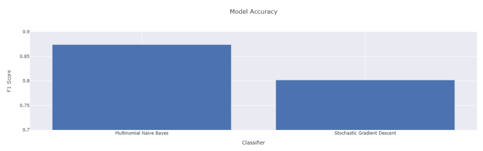


Figure 11: Graph displaying F1 scores for MNB and SGD

The F1 scores shown above display an above-average SGD model, and a good MNB model.

4 Conclusion

In conclusion, the presented approach adeptly tackles the issue of misinformation dissemination on social media platforms, particularly in the aftermath of impactful events. The utilization of the MediaEval 2015 "verifying multimedia use" dataset facilitates a nuanced exploration of linguistic patterns within tweets, enabling the formulation of a robust binary classification system for distinguishing between 'fake' and 'real' social media posts. The comprehensive pre-processing steps, including data cleaning, duplicate handling, and language filtering, contribute to refining the dataset for effective model training. The chosen machine learning algorithms, Multinomial Naive Bayes (MNB) and Stochastic Gradient Descent (SGD) prove suitable for text classification tasks, showcasing computational efficiency and memory optimisation. Evaluation metrics, encompassing confusion matrices and F1 scores, demonstrate the models' ability to discern between fake and real posts, with both algorithms exhibiting above-average performances, but MNB being the most efficient for the scenario. This systematic pipeline not only aids in the accurate classification of social media content but also lays the groundwork for future advancements in combating misinformation and upholding the integrity of public discourse on digital platforms.

Some possible future improvements would be to integrate advanced natural language processing techniques for semantic analysis and extend the model to incorporate image processing methods, to provide a more comprehensive understanding of multimedia content. Moreover, ensemble methods, combining predictions from diverse models, and thorough hyperparameter tuning for the algorithms used can optimise overall performance.

References

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

[Géron, 2017] Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

[Islam et al., 2019] Islam, M. Z., Liu, J., Li, J., Liu, L., and Kang, W. (2019). A semantics aware random forest for text classification. page 1061–1070. Association for Computing Machinery.

[McCallum and Nigam, 1998] McCallum, A. and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *AAAI Conference on Artificial Intelligence*.

[Pranckevivcius and Marcinkevivcius, 2017] Pranckevivcius, T. and Marcinkevivcius, V. (2017). Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification. *Balt. J. Mod. Comput.*, 5.

[scikit-learn developers, 2024] scikit-learn developers (2007 - 2024).

[Watson, 2001] Watson, T. J. (2001). An empirical study of the naive bayes classifier.

[Zhou, 2021] Zhou, Z. (2021). *Machine Learning*. Springer.