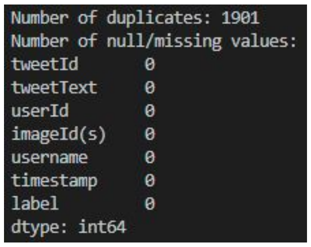
COMP3222 Coursework - MediaEval
January 2023

Ben Hipwell
University of Southampton
bh3g19@soton.ac.uk - bh3g19

1. Introduction

The first analysis of the dataset is via the 'pandas' functions 'info()'. This provides me with a visualisation of the head and foot of the dataset, with 4 examples from each, so that it can be easily interpreted along with the number of rows and columns to give a sense of the database scale. It also provides the data below each of the column names, of which the datatype can be examined for each using 'dtypes()'. See figure: 10 on page 6

To examine the quality of the dataset I check the number of duplicates within the 'tweetText' field and the number pf missing or null values across the entire database. I am only checking the 'tweetText' field as I expect there to be duplicates within other columns, for example the label and image ids. From the results below, there are a reasonably large number of duplicates in the dataset which will therefore mean the dataset is smaller than first thought, potentially affecting the end results. Despite this, there are no missing or null values within any columns of the dataset which implies good quality data and removes an extra step of pre-processing. The functions used to calculate these statistics are also using the 'pandas' Python library: '.duplicated().sum()' and '.isna().sum()'.



```
Number of duplicates: 1901
Number of null/missing values:
tweetId       0
tweetText     0
userId        0
imageId(s)    0
username      0
timestamp     0
label         0
dtype: int64
```

Figure 1. checking duplicates and missing values of the dataset

Another way I have checked the quality and consistency of the dataset is through the variance in the length of the tweets. From this boxplot using the lengths of the tweets, it is clear there are some major outliers that I will remove during pre-processing:

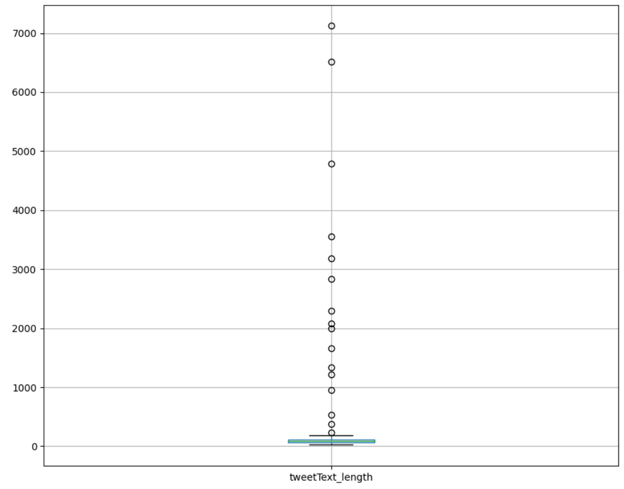
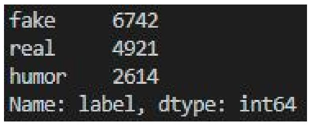


Figure 2. boxplot displaying lengths of the tweets

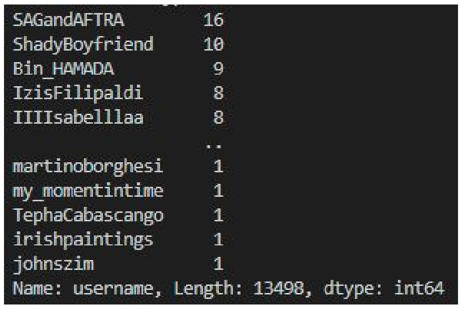
To determine if there is any bias in the dataset, I first check to see the proportion of real tweets to fake/humour tweets. As seen below, it is clear that there are many more fake tweets than real tweets which may cause the classification of fake posts to be of higher accuracy than real posts, bringing the overall F1 score down.



```
fake     6742
real     4921
humor    2614
Name: label, dtype: int64
```

Figure 3. number of fake, humour & real tweets before pre-processing

Additionally, another possible bias could certain users having more tweets than others in the dataset. To check this I have used the function 'value_counts()' to count the number of occurrences of each username, which at displayed some this information:

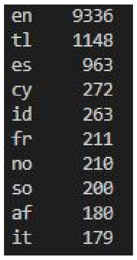


```
SAGandAFTRA        16
ShadyBoyfriend     10
Bin_HAMADA          9
IzisFilipaldi       8
IIIIsabelllaa       8
                   ..
martinoborghesi     1
my_momentintime     1
TephaCabascango     1
irishpaintings      1
johnszim            1
Name: username, Length: 13498, dtype: int64
```

Figure 4. value counts of the username field

This shows that there is an in-balance of the source of the tweets. Even after removing the duplicate rows based on the tweet text, there were still some large disproportions and therefore, reduces the quality of the data.

After some pre-processing, I am also able to check the languages of the text. This will give me an idea into how I want to approach the tweets of varying languages to get the best F1 score possible. Here are the counts of the top 10 languages in the dataset:



```
en    9336
tl    1148
es     963
cy     272
id     263
fr     211
no     210
so     200
af     180
it     179
```

Figure 5. top 10 most used languages

2. Algorithm Design

2.1. Pre-processing

In the first stage of pre-processing I have changed all of the occurrences of the 'humour' label to a 'fake' label in line with the classification goals of this task between only 'real' and 'fake', with humour classifying as 'fake'. Additionally, to classify data in machine learning, the classes are required to be numerical instead of strings. Therefore, I have mapped all of the 'fake' labels to 0 and all of the 'real' labels to 1 so that they can be used in the classification stage.

The next step of pre-processing is the removal of tweets that are over certain length. For this value, I have chosen 350, which seems reasonable after reviewing the boxplots. I have also made sure to carry this out before any of the other text-preprocessing to save time that would be spent manipulating very large tweets that are going to be removed. Along with tweets that are too long, I have also removed tweets that, after pre-processing, are less than 4 words long to remove unimportant tweets and to avoid the issue of not being able to detect the language of a tweet because there are not enough words. Getting to this value, removing words of length 2 reduced the F1 score, length 3 increased the F1 score slightly, length 4 increased it the most and length 5 began to decrease it again. Here is the updated box plot after having removed the outlying tweets:

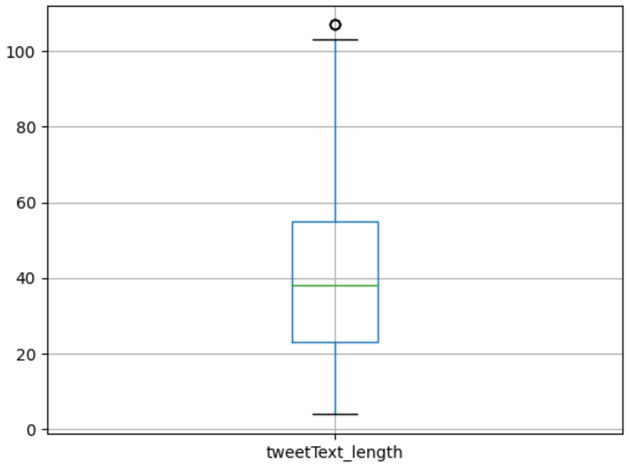


Figure 6. tweet length boxplot after pre-processing

Moving onto the pre-processing of the tweet text, I firstly remove any twitter handles by removing any words starting with '@'. This is especially important as I have noticed duplicate tweets, with the only difference being the twitter handle, therefore after this change they will all be removed. I then remove any urls in the tweet by looking for words starting with 'http' and convert all the text to lower case before tokenizing the text, with the Natural Language Toolkit library, into a set of words for further processing. For the last part of this pre-processing, I remove any words that are less than 4 characters long, as this saw a significant increase in the F1 score. [1] [2]

Next, I move onto lemmatization to normalize the text so that it is ready for further processing. This is the process of reducing a word to its base form, however unlike stemming, it goes a step further and extends this form to its intended meaning. Even though it requires more computation than stemming, it does not take too long to run on this dataset and provides a higher F1 score. I have also experimented with the implementation of both, however this also results in a lower F1 score than just lemmatization on its own. [3]

Another few pre-processing steps that I tried include removing all characters outside of a-z and A-Z and removing all words that were not nouns or adjectives, however both of these slightly lowered the final F1 score. Another unexpected pre-processing investigation was the removal of emojis. Removing emojis from the text reduced the F1 score, of which I can only hypothesis this is down to the likelihood of fake tweets having a higher chance of containing emojis.

In terms of the languages of the text, I have decided to not translate or exclude any tweets that were not in English. This is because the process of detecting and/or translating a language is very time consuming, and it did not bring any benefits to the F1 score on the test data set.

The final part of my pre-processing is the removal of duplicate rows. This will help reduce the bias within the data and prevent any extra over-fitting of data to the duplicated texts. To carry this out I used the 'pandas' library function 'drop_duplicates', specifying the tweet text field. I have carried this out after the pre-processing of the tweet text because it will now remove very similar tweets that just simplify down the same key words. Additionally, I have removed the tweets from the 4 most occurring usernames to prevent them from skewing the training data, of which the value 4 has been chosen as this increased the F1 score the most.

2.2. Feature Selection & Dimensionality Reduction

The most important aspect of this task has revolved around a TF-IDF vectorizer. This is a popular method of extracting features from text in a numerical format so that it is ready for classification. To use this, I have used the 'TfidfVectorizer' within sklearn to weight terms within the tweet text based on how important they are. The main two parameters that I began tweaking were the maximum number of features and the workings of the feature selection via ngrams. In terms of ngrams, I started with creating features based on selecting individual words (unigram) and then onto individual words and a series of two words (unigram and bigram) which increased the F1 score. Changing the 'analyzer' from creating features from words to characters opened up new possibilities of features. My final parameter for ngram range to gain the best possible F1 score became (3,5), which means it will create a sparse matrix of 3-grams, 4-grams and 5-grams (3,4 and 5 series of characters). For the maximum number of features, I settled on 40,000, which seems like a very large number, however, it is the value which provided the best F1 score. [2] [4] [5]

Another feature that I incorporated was the sentiment of the tweet. I completed this using the NLTK feature known as VADER to complete this, providing a score between -1 and 1, with 0 being neutral. To use this as a feature, I had to add 1 to every score so that the range was instead between 0 and 2, with 1 being neutral as the classifier cannot handle negative values. My original idea was to use a BERT algorithm for NLP, however having seen an only slight improvement of the F1 score after adding this as a feature, I thought it would be out of scope of this project. I experimented with the different outputs of the sentiment analysis, where using the compound value of positivity and negativity performed the best. The main reason there is only a slight increase in score would be down to most of the tweets being very neutral as seen in this graph [6]:

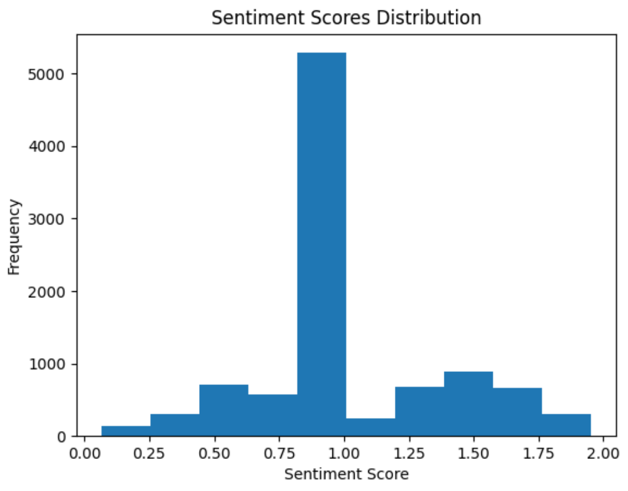


Figure 7. sentiment distribution, scaled from 0 to 2, with 1 being neutral

The last set of features that I have been adding and disabling throughout my iterative development is the implementation of K-means clustering. This is an unsupervised machine learning algorithm aimed to group the data into a specified number of clusters based on similar characteristics. This feature is quite computationally expensive and also produces slightly fluctuating F1 scores, and therefore has been disabled during testing and tweaking of other aspects of my program. When added as a separate feature there can sometimes be an increase in the F1 score by around 0.01, however there are sometimes loses of around 0.02, and therefore it is unlikely that I will continue to use it. [7] [5]

Additionally, another method of dimensionality reduction via feature selection is using the select k best approach. Essentially, this selects the features that contribute the most, where I have chosen to select the 4000 most important features, reducing the input features from the text alone from 40,000 to 4,000. In doing this, it has brought the time taken to train and validate the model from around 2.6 seconds to

0.2 seconds, measured using 'time.process_time()', whilst only lowering the F1 score by under 0.003.

Furthermore, there have been many other features that I have added to try and increase the F1 score that have not proven successful. This includes using the hour of the day that the tweet was posted in the aims of seeing a pattern between the times of the day real and fake tweets are posted and the username of the author of the tweet after preprocessing of the username text. Both of these features were unsuccessful and only reduced the F1 score.

2.3. Classification

The first types of classification methods that I attempted were ensemble approaches. This included XGBoost, and later Random Forest, of which did not provide particularly good results for the F1 score. After extensive research, it seemed as though Naive Bayes classifiers were more popular with NLP tasks such as this. [2] [5]

3. Evaluation

Due to the iterative, approach I took during development, I have explained throughout this report how each part of my design has been adapted to improve the overall F1 score instead of combining it all within this section, of which will focus more on the classification side of the evaluation.

These are the results of differing Naive Bayes algorithms available within sklearn below. When calculating these, I ensured clustering was not being performed so that they can be compared fairly as clustering provides fluctuating results. When using the 'Select K Best' features approach, I tweaked the value of K for each classifier used. The most I gained from this was 0.1 increase in the F1 score with the Bernoulli NB by changing K to 5000, however the rest of the classifiers saw diminishing results when changing it from 4000.

As seen in the tables below, despite the Select K Best features approach reducing the training time for the classifiers, it actually increases the overall computation time of the program. I assume this is due to the time taken to decide which are the best features to select.

With Select K Best (3 s.f)						
NB Classifier	F1 Score	Time Elapsed (s)	TP	FP	TN	FN
Multinomial	**0.861**	17.1	981	89	2457	228
Gaussian	0.827	17.5	1016	233	2313	193
Complement	0.714	16.9	1111	791	1755	98
Bernoulli	0.703	17.4	1090	803	1743	119

Without Select K Best (3 s.f)						
NB Classifier	F1 Score	Time Elapsed (s)	TP	FP	TN	FN
Multinomial	**0.863**	13	1008	118	2428	201
Gaussian	0.07	19.5	74	859	1687	1135
Complement	0.797	13.3	1109	466	2080	100
Bernoulli	0.616	16	1099	1261	1285	110

One interesting insight into these results is the very poor performance of the Gaussian NB when there has been no dimensionality reduction, suggesting its poor performance at classifying a large number of features. Despite this, lowering K below 4000 saw no gain in the F1 score.

On the other hand, I investigated the performance of a Support Vector Machine to classify the data. Keeping the default parameters, and using the Select K Best approach, the first run took 130.1 seconds, and provided an F1 score of 0.801. Fine-tuning the parameters, e.g. changing the kernel, gamma and C values, provided no gains in F1 score, where changing the 'Select K Best' features value to 3000 saw only a gain of 0.007 and therefore I stopped the evaluation of the SVM, especially with the high computation time.

Finally, re-investigating the use of ensemble learning through the XGBoost classifier, the use of the 'Select K Best' dramatically increased the F1 score with the default parameters to 0.783 and then to 0.786 after fine-tuning K to 2000. From there, fine-tuning the classifier parameters so that 'max_depth' is 5, and 'subsample' & 'colsample_bytree' are both 0.9 saw the F1 score increase to 0.81. Despite this improvement from the XGBoost model from when I first experimented with it during development, it is still outperformed by the Multinomial Naive Bayes classifier.

Here are two graphs comparing the best fine-tuned versions of my implementation for each of the 6 classifiers that I experimented with:

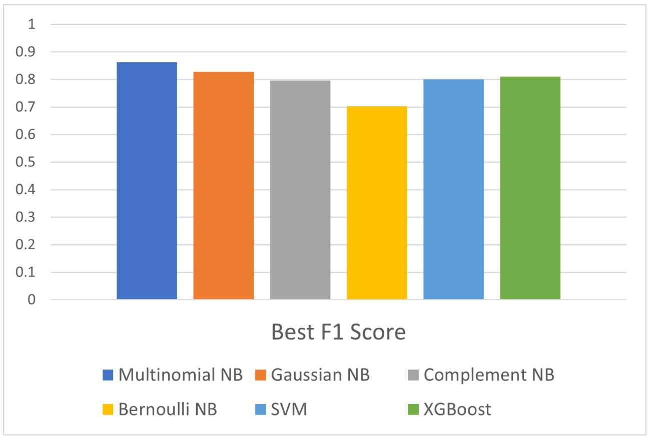


Figure 8. simple graph comparing f1 scores of the classifiers

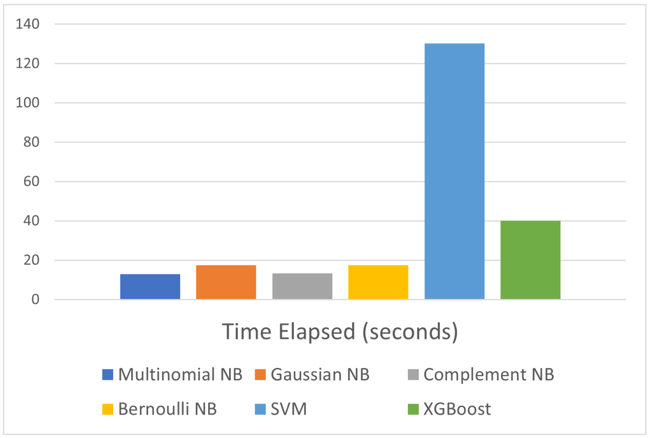


Figure 9. simple graph comparing time elapsed of the classifiers

4. Conclusion

To conclude, my approach to this task through the use of my dataset investigation, data pre-processing, feature selection, dimensionality reduction and classification has been successful, gaining a maximum F1 score of 0.863. This was possible through the version of my model that used TF-IDF to extract numerical features in the pre-processing tweet text, before being reduced into a total of 4000 features and used to train a Multinomial Naive Bayes classifier.

4.1. Future Improvements & Lessons Learnt

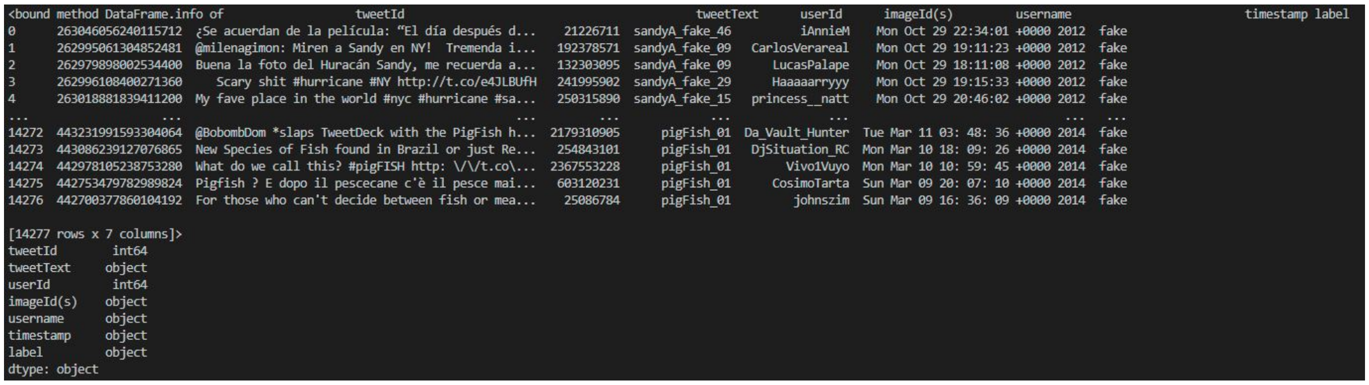
Firstly, a large possible improvement over the commonly used TF-IDF Vectorizer is the use of the Word2vec model. This model supposedly is more efficient than TF-IDF as it extracts much fewer features (100-300) and performs better on average by a significant amount. [8] In this research by Joseph Lilleberg, Yun Zhu & Yanqing Zhang (2015), they concluded that the use of Word2vec weighted by TF-IDF without stop words followed by TF-IDF without stop words resulted in the best accuracy. [9]

Additionally, the powerful BERT model can be used for text classification. This pre-trained deep learning model is trained on a huge amount of data and is versatile to perform a range of complex NLP tasks. This example of transfer learning can be fine-tuned using custom data so that it can adapt to specified types of language if needed. In a report by Gonzalez-Carvajal & Garrido-Merchan (2020), they compared using the BERT model for text classification to traditional approaches such as TF-IDF. One of their experiments involved classifying real or fake tweets about a real disaster, similar to this project, where their use of the BERT model outperformed the traditional competition. [10] Including its powerful text classification, it can also be used for sentiment analysis, where outperforming my approach using NLTK's VADER model could also provide an improved F1 score.

Using Word2vec and the BERT model, I believe there is much more room for improvement over my more traditional approach, whether that be using just one of these methods, or a combination of both. From there, re-testing and re-tweaking classifiers that work best with their outputs would be the next step to obtaining an improved F1 score.

Finally, the use of automated parameter fine-tuning would be a technique I would use if I were to tackle this challenge again or attempt something similar. This would save time when it comes to manually changing parameters and comparing results. To do this, a tool such as GridSearchCV within sklearn could be used.

5. Extra Images



```
<bound method DataFrame.info of                  tweetId                                          tweetText      userId      imageId(s)        username                     timestamp label
0      263046056240115712  ¿Se acuerdan de la película: “El día después d...    21226711  sandyA_fake_46         iAnnieM   Mon Oct 29 22:34:01 +0000 2012  fake
1      262995061304852481  @milenagimon: Miren a Sandy en NY!  Tremenda i...   192378571  sandyA_fake_09   CarlosVerareal   Mon Oct 29 19:11:23 +0000 2012  fake
2      262979898002534400  Buena la foto del Huracán Sandy, me recuerda a...   132303095  sandyA_fake_09      LucasPalape   Mon Oct 29 18:11:08 +0000 2012  fake
3      262996108400271360     Scary shit #hurricane #NY http://t.co/e4JLBUfH   241995902  sandyA_fake_29      Haaaaarryyy   Mon Oct 29 19:15:33 +0000 2012  fake
4      263018881839411200  My fave place in the world #nyc #hurricane #sa...   250315890  sandyA_fake_15   princess__natt   Mon Oct 29 20:46:02 +0000 2012  fake
...                   ...                                                ...         ...             ...              ...                              ...   ...
14272  443231991593304064  @BobombDom *slaps TweetDeck with the PigFish h...  2179310905      pigFish_01  Da_Vault_Hunter  Tue Mar 11 03: 48: 36 +0000 2014  fake
14273  443086239127076865  New Species of Fish found in Brazil or just Re...   254843101      pigFish_01   DjSituation_RC  Mon Mar 10 18: 09: 26 +0000 2014  fake
14274  442978105238753280  What do we call this? #pigFISH http: \/\/t.co\...  2367553228      pigFish_01        Vivo1Vuyo  Mon Mar 10 10: 59: 45 +0000 2014  fake
14275  442753479782989824  Pigfish ? E dopo il pescecane c'è il pesce mai...   603120231      pigFish_01      CosimoTarta  Sun Mar 09 20: 07: 10 +0000 2014  fake
14276  442700377860104192  For those who can't decide between fish or mea...    25086784      pigFish_01         johnszim  Sun Mar 09 16: 36: 09 +0000 2014  fake

[14277 rows x 7 columns]>
tweetId       int64
tweetText    object
userId        int64
imageId(s)   object
username     object
timestamp    object
label        object
dtype: object
```

Figure 10. info() and dtypes() of the dataset

References

[1] Tajinder Singh and Madhu Kumari. Role of text pre-processing in twitter sentiment analysis. *Procedia Computer Science*, 89:549–554, 2016. Twelfth International Conference on Communication Networks, ICCN 2016, August 19–21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.

[2] Abdullah-All-Tanvir, Ehesas Mia Mahir, Saima Akhter, and Mohammad Rezwanul Huq. Detecting fake news using machine learning and deep learning algorithms. pages 1–5, 2019.

[3] Divya Khyani, BS Siddhartha, NM Niveditha, and BM Divya. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 2021.

[4] Seyyed Mohammad Hossein Dadgar, Mohammad Shirzad Araghi, and Morteza Mastery Farahani. A novel text mining approach based on tf-idf and support vector machine for news classification. pages 112–116, 2016.

[5] Wenlin Han and Varshil Mehta. Fake news detection in social networks using machine learning and deep learning: Performance evaluation. pages 375–380, 2019.

[6] Amrita Shelar and Ching-Yu Huang. Sentiment analysis of twitter data. pages 1301–1302, 2018.

[7] Khumaisa Nur'aini, Ibtisami Najahaty, Lina Hidayati, Hendri Murfi, and Siti Nurrohmah. Combination of singular value decomposition and k-means clustering methods for topic detection on twitter. pages 123–128, 2015.

[8] Cai-zhi Liu, Yan-xiu Sheng, Zhi-qiang Wei, and Yong-Quan Yang. Research of text classification based on improved tf-idf algorithm. pages 218–222, 2018.

[9] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. Support vector machines and word2vec for text classification with semantic features. pages 136–140, 2015.

[10] Santiago González-Carvajal and Eduardo C. Garrido-Merchán. Comparing bert against traditional machine learning text classification, 2020.