COMP3222 Mediaeval Text Classification CourseWork Report

Chenyang Wu 32588925

January 2024

Contents

1 Introduction

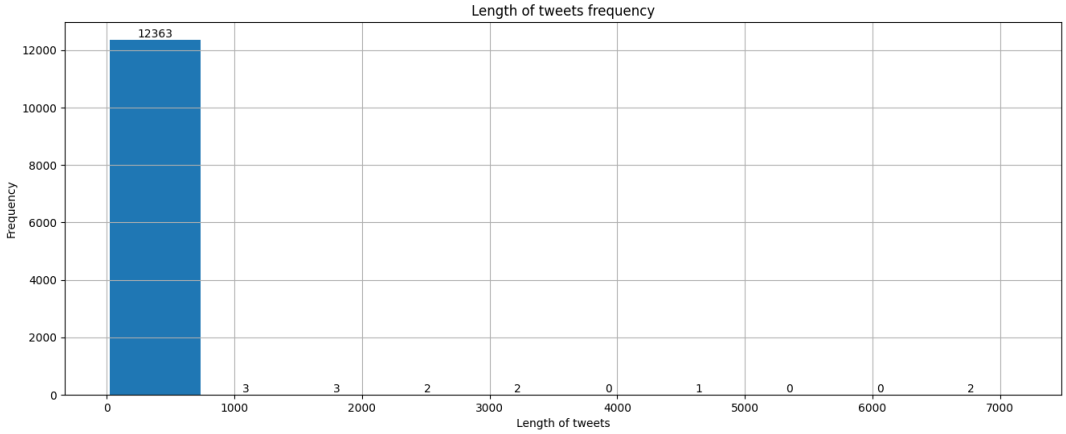
In this project, I aim to train machine learning algorithms to verify whether a tweet is fake or real with the provided datasets. The aim is to achieve an acceptable F1 score by tuning the chosen algorithms. I aim to implement 2 algorithms after preprocessing the datasets based on their structure. There are two classes in this data set, fake/humour and real which lead to this coursework as a binary classification problem.

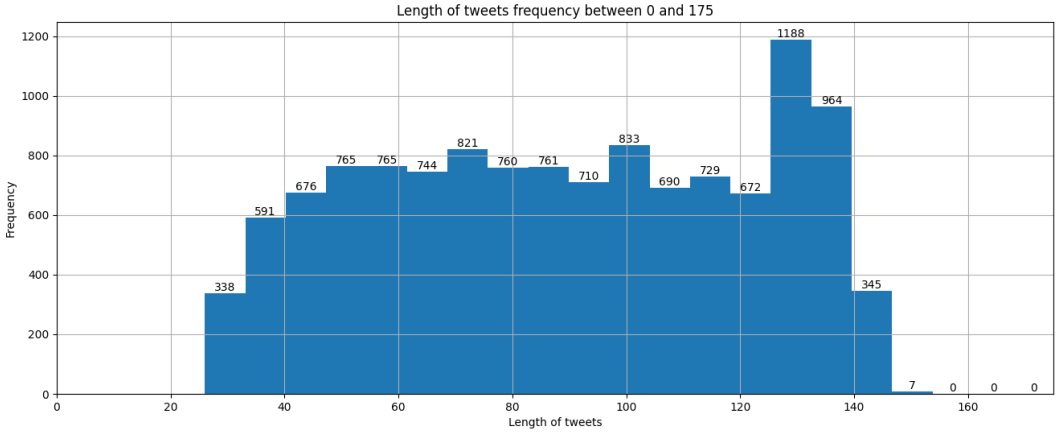
I will mainly use this dataset's "tweetText" and "label" to train my chosen algorithm. The training set and Testing set are already separated. For data preprocessing, I plan to focus on "tweetText", extracting possibly sufficient information from "tweetText" of the dataset and training the algorithm with the label. This will be a supervised machine learning.

2 Data Analysis & Data Pipeline design

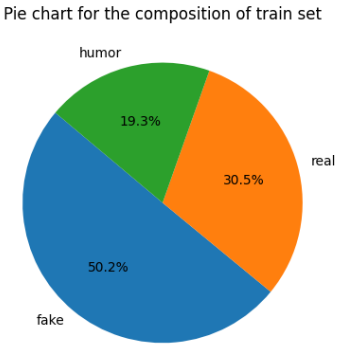
There are 14276 records in the training set. With the help of Python packages like *pandas, numpy, matplotlib, nltk* I analysed and preprocessed the training data as follows[2].

First, I converted both train and test sets from ".txt" files into ".csv" format and loaded them into *pandas DataFrame* for future data analysis. I will mainly focus on the datasets' "tweetText" and "label" columns. I created new *pandas DataFrame* "trainSet" and "testSet" with only the "tweetText" and "label" columns of the original datasets. Then I found that there are only 12376 records of unique data. So I chose to drop the duplicates in the data sets, 12376 entries remain in the data set. I then looked at the length of tweets, finding that they have lengths mainly between 0 and 175 characters with a longest of about 7000 characters as figures shown below.

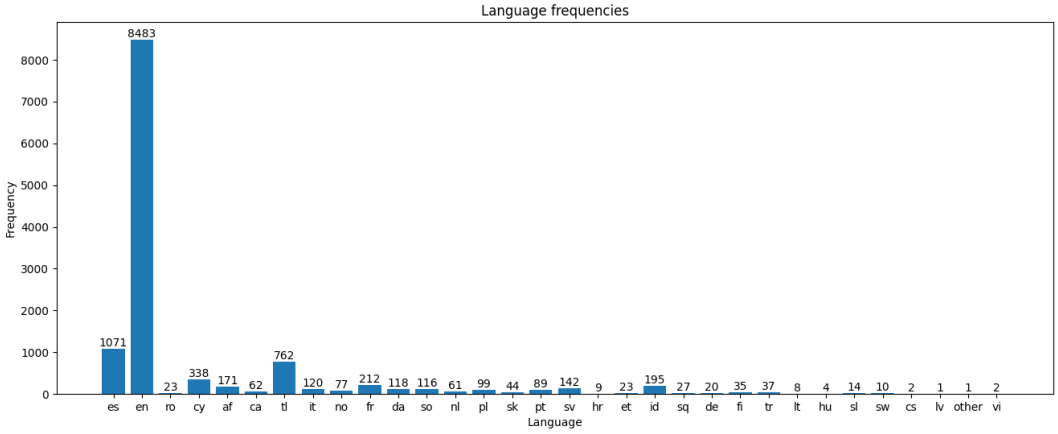




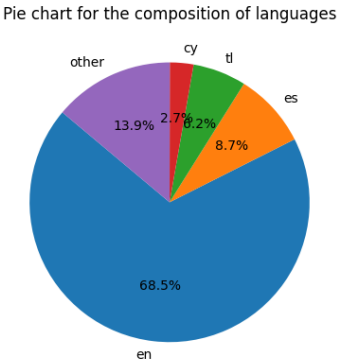
Also by plotting a pie chart, I got the composition of labels in the train set as below. There are 19.3% "humor" and 50.2% "fake" which makes the total number of fake posts in the training set 69.5% of the set, the remaining 30.5% are real posts.



After looking at some records of the "tweetText" column, I found that there are noises like emojis, stopwords, and URLs throughout the dataset. Also, there are "tweetText" in languages other than English. So I applied functions to remove the noises from the original texts, detected their languages, and appended their languages in a new column of my data frame. Based on the frequency of languages, I plotted the following graphs.

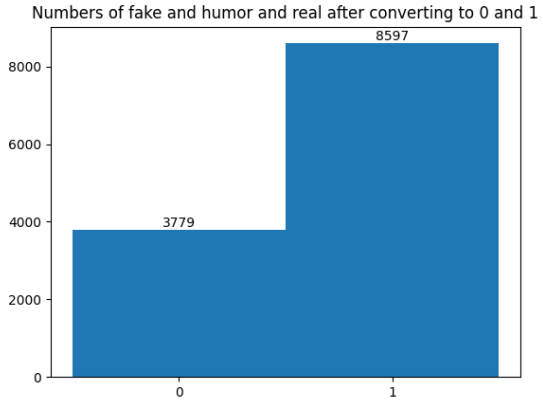


and a pie chart which groups any language that appears less than 300 times into an "other" subset. It shows that 74% of the posts were in English, 8.4% were in Spanish, 4.4% were in Tagalog and the remaining 13.1% were in other languages.

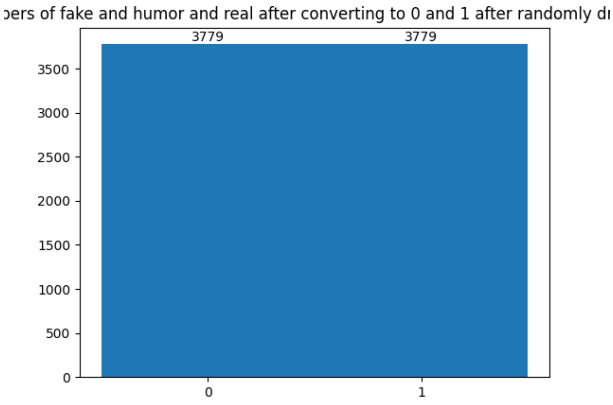


Apart from the plotting, I put all the cleaned text into a "removed" column in my data frame. However, my stopwords removing function can only pick up English stopwords, so I then translated all text into English and performed text cleaning on the translated text again for accuracy. As I will be using English as the only language to train my model, I translated non-English text from the testing set and also performed a text cleaning, the same as I did to the training set. The text cleaning procedure includes "URL removal", "stopwords removal", "lowering all letters", "non-alphanumeric characters removal" and "emoji and punctuation removal".

There are 3 labels "fake", "humor" and "real". To make the model pick up the label easily, I changed all labels in both train and test sets into 0 and 1, 1 for "fake" and "humor", and 0 for "real". After converting, I plotted a bar plot with this initial analysed training data as below.

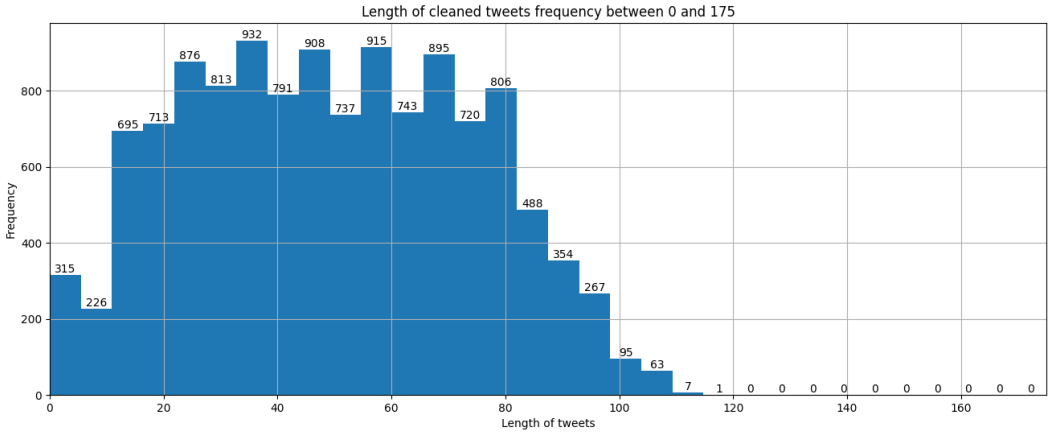


From here I found that the training data is imbalanced with the number of "1" labels more than 2 times the number of "0" labels, so I randomly dropped the entries with "1" labels by the number difference to balance the training set just in case that I need this structure of training set later for training.



Before implementing the algorithms, I sorted the texts into train sets and test sets namely "X_train, y_train" and "X_test, y_test". To prevent overfitting due to outliers, I excluded tweet texts with lengths

greater than 120 characters based on the figure below.



3 Algorithms and Evaluation

As this is a text classification task, after researching papers and surveys through the internet, I started implementing algorithms and training my models[3]. Before fitting the training text data into pipelines, I should first vectorize my training texts. I chose to use *TfidfVectorizer* from *sklearn* and instead of vectorizing the data on their own, I will be including this vectorizer in my pipeline. *TfidfVectorizer* shows that the frequency of a term/token appears in a text, if one token has a high frequency then the weight will be really low as it holds very little power. I will also be using *Pipeline* from *sklearn* to wrap vectorization together with the algorithm.

To calculate the f1 score and evaluate my pipelines, I will mainly use *classification_report*, *f1_score* and *ConfusionMatrixDisplay* from *sklearn* to visualise and analyse the performance of my model.

3.1 Algorithm 1

The first algorithm I chose was *MultinomialNB* from *sklearn.naive_bayes*[3]. This algorithm is good for text classification as it is a probability-based algorithm based on Baye's theorem, it is efficient with both large and small sample sizes and it is easy to implement and fast to run.

I implemented my pipeline as below:

```
mnb_model = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('mnb', MultinomialNB())
])
```

By fitting the training set to this model and asking the model to predict the label for the testing set. I managed to achieve a f1 score of 0.85. The classification report of this model is shown below:

```
              precision    recall  f1-score   support

           0       0.77      0.75      0.76      1209
           1       0.88      0.89      0.89      2546

    accuracy                           0.85      3755
   macro avg       0.83      0.82      0.82      3755
weighted avg       0.85      0.85      0.85      3755

The f1 score for MultinomialNB is:  0.847
```

I can see that 1(fake) has a higher accuracy than 0(real) label. This is probably because of the imbalance number of training data I fit in the model. However I tried using the balanced data where I prepared, and the accuracy was indeed significantly lower than the model trained with the full training set because the size of the data is much smaller. So I will not be using the dropped trainSet in my following implementation to train my model.

With the help of *GridSearchCV* from *sklearn.model_selection* I then performed a grid search on the trained MultinomialNB model and tried to achieve a higher f1 score by tunning the following parameters: "tfidf__max_df" and "mnb__alpha". Where the first parameter "max_df" controls the maximum document frequency in the texts and "alpha" sets the alpha parameter for Laplace smoothing[4][6].

```
param_grid = {
    'tfidf__max_df': [0.5, 0.75, 1.0],
    'mnb__alpha': [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000],
    }
```

The result shows the model starts to overfit. From the pictures below we can see that the model can predict the training set very well but a completely different performance on the testing set.

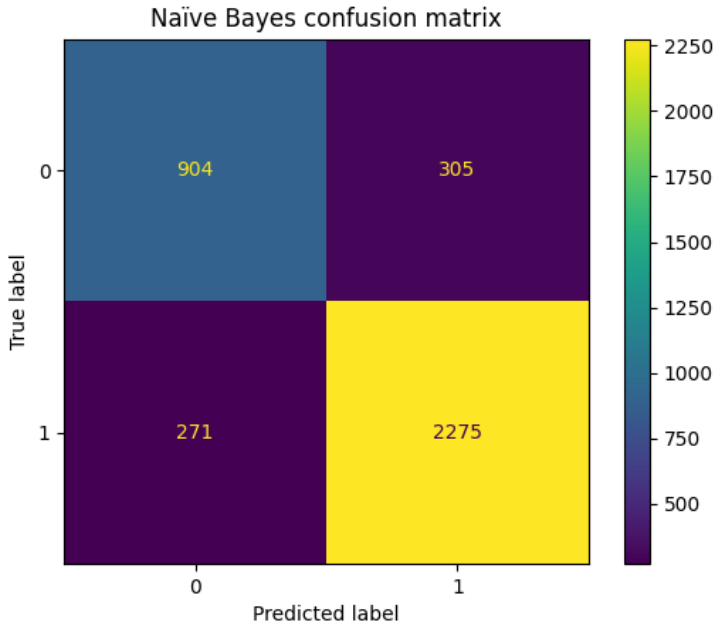
	precision	recall	f1-score	support
0	0.38	0.78	0.51	1209
1	0.78	0.39	0.52	2546
accuracy			0.51	3755
macro avg	0.58	0.58	0.51	3755
weighted avg	0.65	0.51	0.51	3755

The f1 score for MultinomialNB after grid search on testing set is: 0.512

	precision	recall	f1-score	support
0	0.98	0.86	0.91	3774
1	0.94	0.99	0.97	8585
accuracy			0.95	12359
macro avg	0.96	0.92	0.94	12359
weighted avg	0.95	0.95	0.95	12359

The f1 score for MultinomialNB after grid search on training set is: 0.951

So, a 0.85 f1 score is a reasonably high f1 score I could reach with this algorithm. The confusion matrix of this model on the test set is shown below:

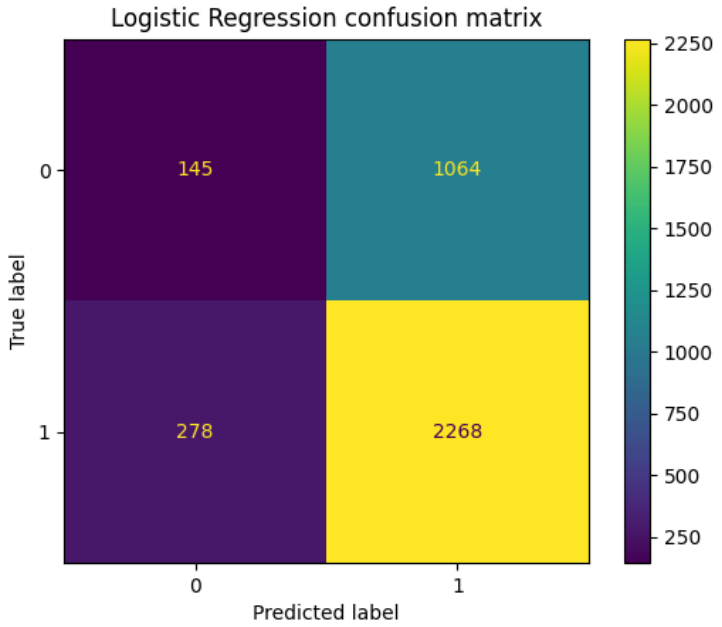


There are 900 true positives, 2247 true negatives, 299 false positives and 260 false negatives[8].

3.2 Algorithm 2

Although we are only asked to implement 2 algorithms, I tried 2 different algorithms, *Logistic Regression* and *Support Vector Machine*, however, the logistic regression algorithm did not perform so well on the data set so I only tried tunning the support vector machine model.

I first implemented *Logistic Regression* from *sklearn.linear_model* together with *TfidfVectorizer*. [7]This model with the default settings only scored a f1-score of 0.64 on the test set with 0.17 on the real label and 0.77 on the fake label. This is caused by the imbalance of the training data and the confusion matrix is shown below.



There are 140 true positives, 2236 true negatives, 1059 false positives and 271 false negatives[8]. As a result, I chose to implement another model which is *SVC* from *sklearnṡvm* with *TfidfVectorizer* and setting SVC's probability parameter to "True" and kernel set to "linear"[1].

```
# Setting up a pipeline for SVM classification with
pipeline_svm_dropped = Pipeline([
    ('tfidf', TfidfVectorizer()),  # Convert text da
    ('svm', SVC(probability=True, kernel="linear"))
])
```

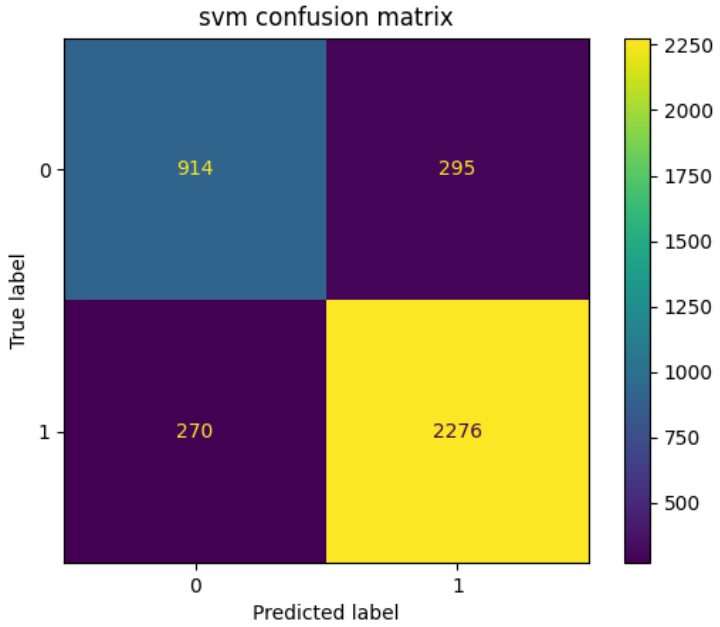
This time the model shows a better performance of 0.85 on the f1-score.

```
The f1 score for Support Vector machine is:  0.85
              precision    recall  f1-score   support

           0       0.77      0.76      0.76      1209
           1       0.89      0.89      0.89      2546

    accuracy                           0.85      3755
   macro avg       0.83      0.82      0.83      3755
weighted avg       0.85      0.85      0.85      3755
```

We can see from the classification report that this SVM model performs similarly to the MultinomialNB model I used in Algorithm 1 (3.1). The model is still performing better on 1(fake) label due to the data imbalance. Similarly, this SVM model scored a much lower score with the balanced data due to the size being too small. I then performed a grid search with the following parameters[5][6]:

```
param_grid = {
    'tfidf__max_df': [0.5, 0.75, 1.0],
    'tfidf__min_df': [1, 2, 3],
    'svm__C': [0.1, 1.0, 10],
}
```

The grid search results are the same as the original model however this time the model does not overfit and the confusion matrix is as below:



There are 916 true positives, 2282 true negatives, 293 false positives and 264 false negatives[8].

3.3 Further Adjustments

After fitting the models with my original modified training data, I also fitted them with some further modifications. I appended "image ID" towards the back of the cleaned tweet texts and fit them to both models. And the score both models achieved did not differ from the previous result too much as shown in the classification report below.

The f1 score for MultinomialNB with imageid is: 0.83

	precision	recall	f1-score	support
0	0.73	0.75	0.74	1209
1	0.88	0.87	0.87	2546
accuracy			0.83	3755
macro avg	0.80	0.81	0.81	3755
weighted avg	0.83	0.83	0.83	3755

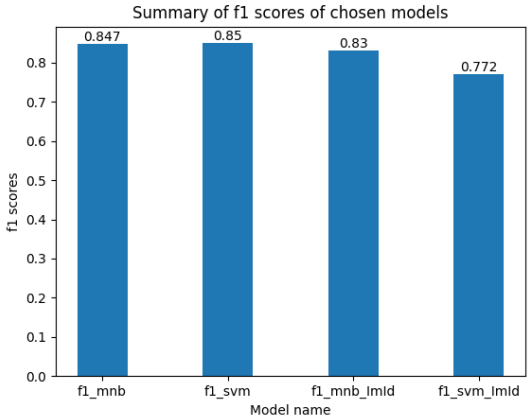
The f1 score for Support Vector machine with imageid is: 0.772

	precision	recall	f1-score	support
0	0.62	0.78	0.69	1209
1	0.88	0.77	0.82	2546
accuracy			0.77	3755
macro avg	0.75	0.77	0.75	3755
weighted avg	0.79	0.77	0.78	3755

Accuracy: 0.7720372836218375

4 Conclusion

Here is a plot of the performance of all models I trained in this coursework:



We can see from the plot that the Support Vector Machine Algorithm performs the best with this data set scoring 0.85 on f1-scores followed by MultinomialNB Algorithm from Naive Bayes scoring 0.847 on f1-scores. If time allows I would perform a larger grid search on more parameters so that can get a more precise result on parameter tuning. On the other hand data preprocessing is also very important for machine learning. Even though I did not manage to get higher f1-scores by adding the "Image ID" to the back of cleaned "tweet texts", I still believe that with more modification and probably a larger training data set, I will be able to achieve a much higher f1 score in this project. However, due to the limitation of training data and time limitations, the above scores are what I achieved for this project.

References

[1] Fabrice Colas and Pavel Brazdil, *Comparison of svm and some older classification algorithms in text classification tasks*, IFIP International Conference on Artificial Intelligence in Theory and Practice, Springer, 2006, pp. 169–178.

[2] Andrea Esuli and Fabrizio Sebastiani, *Training data cleaning for text classification*, Conference on the Theory of Information Retrieval, Springer, 2009, pp. 29–41.

[3] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown, *Text classification algorithms: A survey*, Information **10** (2019), no. 4, 150.

[4] scikit learn, *Multinomialnb parameters*, 2023, `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html` [Accessed: 2024-1-1].

[5] _____, *Svc parameters*, 2023, `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html` [Accessed: 2024-1-1].

[6] _____, *Tfidfvectorizer parameters*, 2023, `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html` [Accessed: 2024-1-1].

[7] Kanish Shah, Henil Patel, Devanshi Sanghvi, and Manan Shah, *A comparative analysis of logistic regression, random forest and knn models for the text classification*, Augmented Human Research **5** (2020), 1–16.

[8] James T Townsend, *Theoretical analysis of an alphabetic confusion matrix*, Perception & Psychophysics **9** (1971), 40–50.