A review of supervised classification techniques in R categorizing short text news headlines

MSc Data Science
University of London
Coursework paper
Name: Saurabhkumar Patel
Word count: 4655

ABSTRACT

The modern world generates vast amount of information. One of the forms of information we perceive daily is news through a digital or social media. Such vast amount of news can be difficult to parse through for the category in which the user is interested. Categorization of news articles allows users to focus on the news groups they are interested in and allows publications to provide appropriate news to the right reader group. Categorization of numerous articles of varying size, nature, and language can be difficult if done manually. Digital news, being electronically readable, allows us to use machine learning techniques to classify news. This paper examines models of the Machine learning techniques for news classification based on news headlines. Further, the paper also evaluates performance metrics of such techniques on headlines of articles collected from the HuffPost from January 2017 to May 2018. We observe that most techniques examined in this paper do perform better than a null model but are not very accurate at category prediction just based on the short-text headline. However, Random Forest, among all discussed in the paper, provides the best balance for accuracy vs F1 score. In the end we discuss some of the shortcomings, and future improvements.

1. INTRODUCTION

News classification in different categories is a multi-label classification problem. Here the goal is to try to automatically assign a category based on the headline. With the increasing amount of news data being generated and distributed worldwide, classification of such articles and threads have become increasing important. In this paper we'll work with data collected from the HuffPost (The Huffington Post, n.d.) collected through web scraping and available on Kaggle (Misra, 2018) . The entire data contains data articles dated from January 2012 to May 2018 with a total count of over 200,000, however for this paper we only take data from January 2017 to May 2018 into account.

News articles in general have hundreds of words and there are vast amounts of articles published every day. Once can text classification on the entire article and use clustering methods to classify them to different groups. However, it can be resource intensive to cluster full articles. Most articles are published with a headline. What if we could use the headlines of an article and try to cluster them. Many researchers have tried to explore this varying level of success. It is a much harder question to tackle as most techniques are based on long text clustering and short text contains data sparsity, irregularity, and less occurrence of a word. (Dai, Li, Li, & Li, 2020)

In this paper try to analyse supervised learning classification techniques that can learn on already classified multi-class train set that contains only headlines and predict articles based on a model train on features defined in a vector space. We will use R to program, model, and evaluate all of our techniques. Primarily we use the `tidymodels` package in R to perform all model trainings. Tidy models use `tidyverse` principles and provides an easy-to-use framework to model machine learning algorithms. (Tidymodels, n.d.) In addition, it also provides prebuilt functions to evaluate the models and plot certain attributes. In addition, we'll use other R libraries to clean, explore, and plot findings of our research. We also provide graphs, equations and tables where needed to help the user understand the process.

The paper is a review of various modelling techniques on short text classification thus we have decided the following to be our Null and alternate hypothesis.

$$H_0 = Null\ model\ is\ the\ only\ model\ that\ provides\ the\ best\ model\ performance$$

$$H_1 = We\ manage\ to\ find\ a\ model\ that\ performs\ better\ than\ the\ null\ model$$

Performance above means the prediction performance of a model to predict the news category based on the headline of the news article measured by all the evaluation metrics we list in the paper. For a model to beat the null model it has to perform better on all metrics. A null model will be a base model which is simple and non-informative that classifies most observations to a random class and provides us a starting off point to start our evaluation journey. Null models have been used by researchers of many domains to test model performance compared to a random or best judgment assignment. It carries importance for our exercise here as our dataset is an imbalanced class one.

1.1 Data exploration

Our goal with data exploration is to learn about characteristics of a data set and plan for cleaning certain content and stemming certain features without making any assumptions on the outcome of our analysis. In statistics, data exploration is often referred to as "exploratory data analysis" and contrasts traditional hypothesis testing. (Harald Piringer, Visplore, 2020) The data is formatted in JSON, has 6 columns, and contains 200,853 observations from the year 2012 to May 2018 obtained from HuffPost. But for this paper we will only take observations more recent than January 2017

Columns:

Name	**Description**	**Used in classification**
category	Category into which the article is classified	Y
headline	Headline of the article	Y
author	Article's author	N
link	Website link to the page	N
short_description	A short description of the content of the article	N
date	Publishing date	N

All are saved as character type; even "date". Thus, we did transformation where needed for further exploration.

1.1.1 Date

While the original dataset spans back till 28/01/2012 our subset used in the paper has the oldest article dated on 01/01/2017 and the most recent one is from 2018-05-26. There seems to be a declining trend in the number of articles published every month. For 2018 we only have data till

May. We preferred more recent articles than older ones as learning on more recent data is more representative of the current situation making the algorithm more suitable for predicting upcoming articles. (Chang & Chang, 2008). In the paper we only discuss articles that are more recent than January 2017.

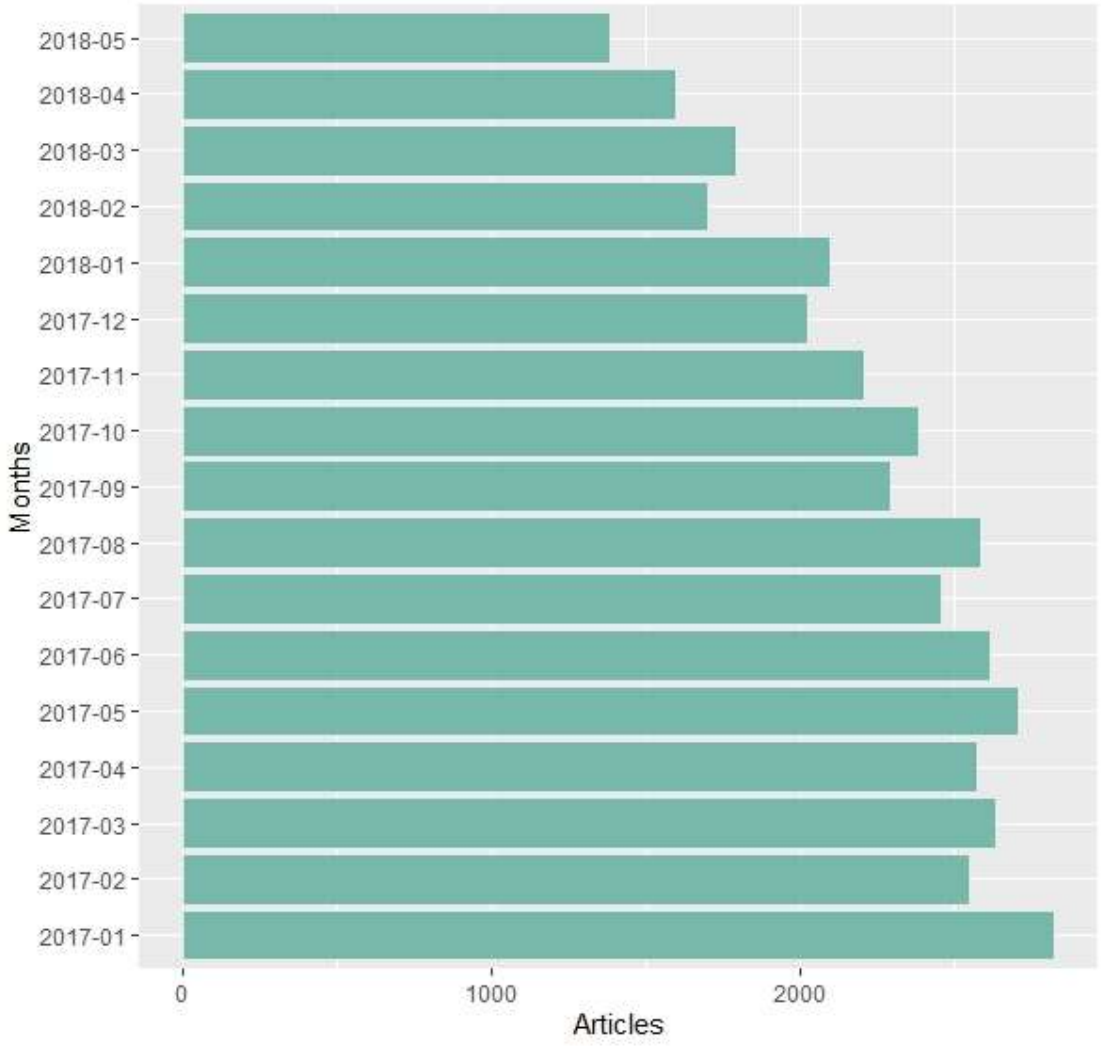

Figure 1: Monthly count for articles from 2017

1.1.2 Category

There are 31 distinct categories that house all the articles of the data set. The distribution among those is however imbalanced. The category with the most articles is *Politics* with 13,680 articles and the one with the least articles is *Fifty* with only 2 articles. We remove all categories with article count below 100, which brings the distinct categories to 26 and the one with the least count is Science with 135 articles. The mean category count comes to 1475.615 while the median to 654.5. This is a substantial finding as the data set is not just multi class but highly imbalanced. This will deter the precision of many classification algorithms even if accuracy is achieved. (Azizi & Reza, 2021)

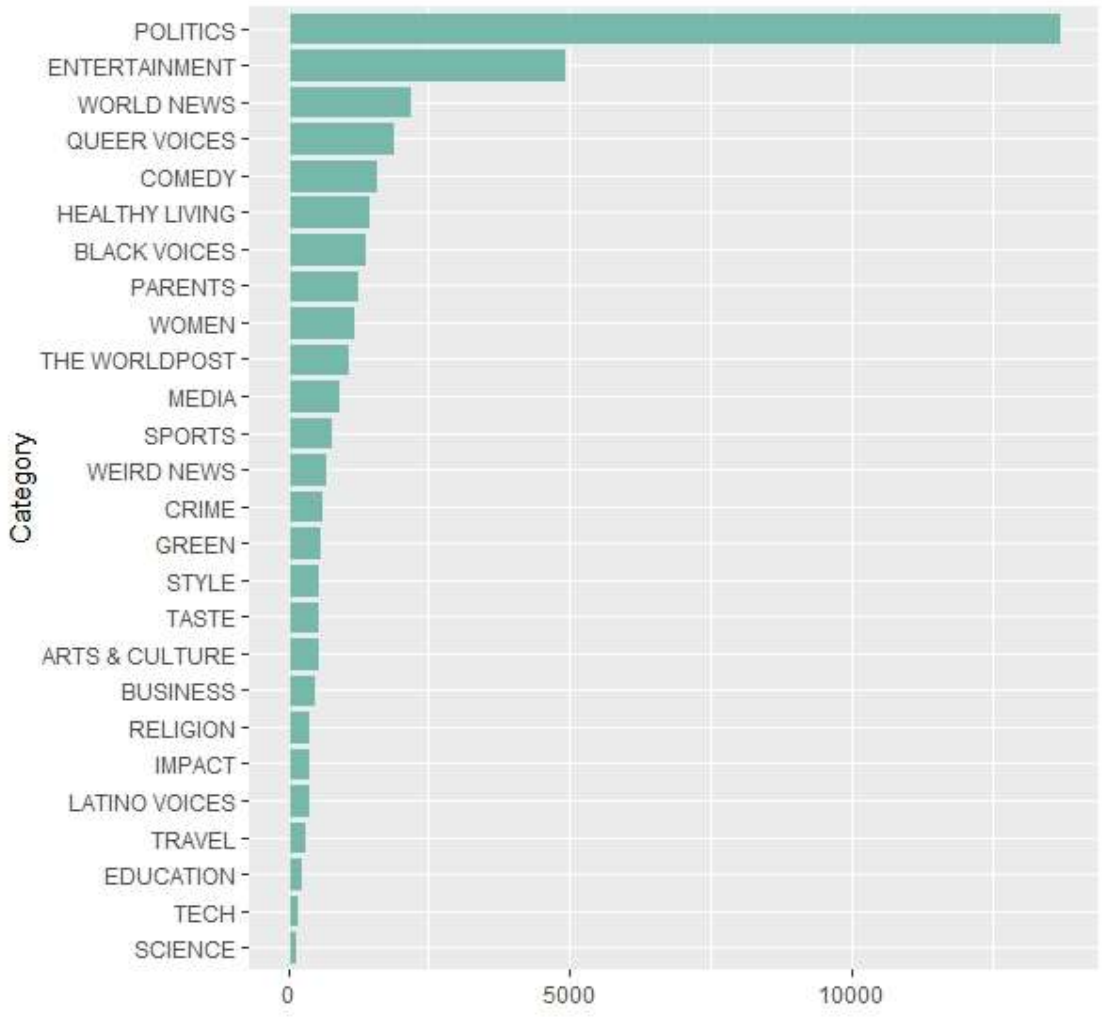

Figure 2: Category count of articles

1.1.3 Headline

The headline contains descriptive text that can be processed to classify articles. The word count of articles ranges from 1 to 27 with a mean of 10.8939 and median of 11 and SD of 2.78. The distribution when plotted looks normally distributed and QQ plot also suggests the same. The main finding here is that there are some data points have only 1 word count. These would need to be treated when pre-processing before the data is used for modelling. This feature is a good candidate for further classification learning. We can use the assigned categories as labels and train the algorithm to predict new articles.

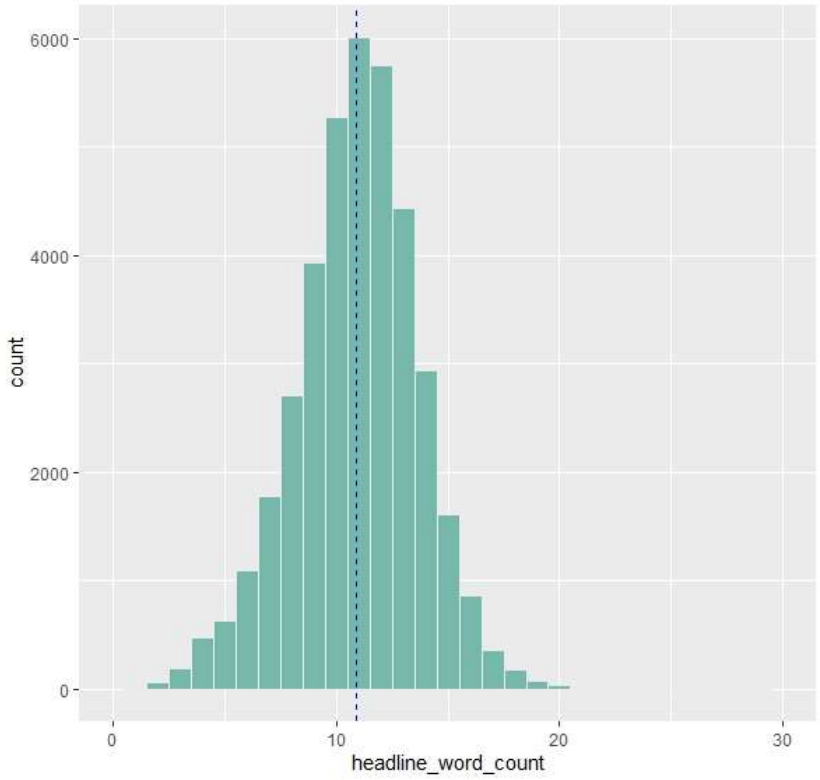

Figure 3: Headline word count distribution

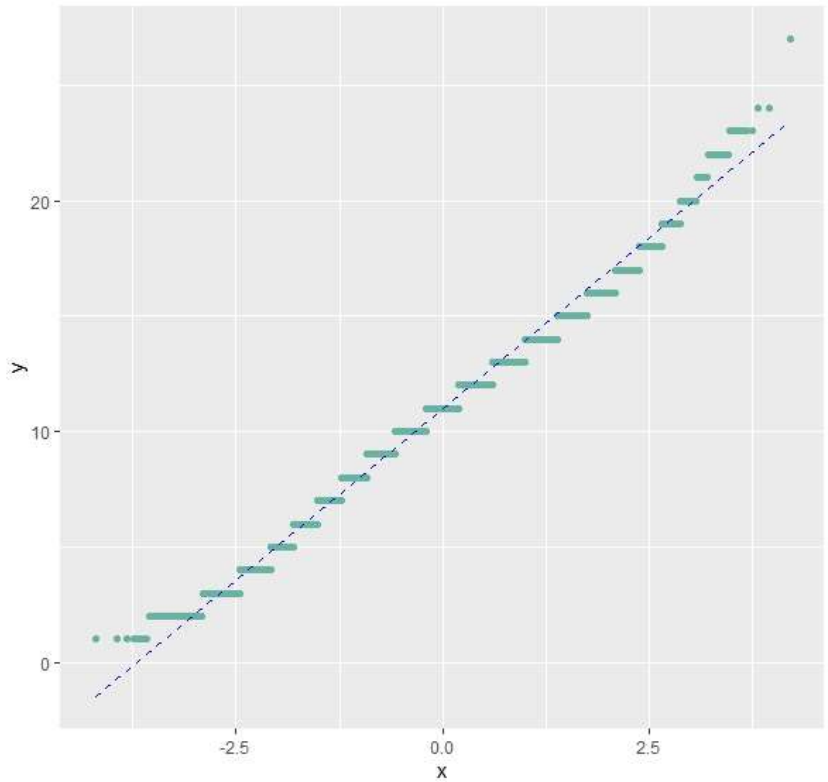

Figure 4: QQPlot for headline word count

1.2 Text pre-processing

1.2.1 Text cleaning:

A normal English sentence contains a variety of what would be considered as noise for machine learning. Wrong spellings, short words, symbols, emojis, punctuations, different cases of written words and others would mostly skew results in a ML algorithm. Thus, it is perfectly fine to remove such words and symbols to have a clean version of the text. In our data set we would deal with capital and title cases and punctuation. We use `tolower()` function to convert all headlines to lower case and then use a regex function to remove all punctuations and unnecessary space, as many punctuations carry a space after its occurrence.

1.2.2 Removing stop words:

Stop words are words used to provide grammatical context in the English language which you'll find in abundance in text strings e.g. for, the, in, to. Depending on the sentence, their occurrence might be scarce or significant. Removal of stop words also helps us reduce the data size and can improve performance. (Shargabi, Olayah, & Romimah, 2011) The `tm` package in `R` comes with a `stopwords` library of 174 words which we would use to filter out of our text with `removeWords` function. Removing them allows further processing to focus on important information.

1.2.3 Lemmatisation

We can group similar forms of the same word together to make it easy process by the algorithm. For example, English word 'walk' can be represented in text in many forms like 'walking', 'walks', 'walked', ect. The process is called Lemmatisation. It helps correctly identify the intended part of the speech and meaning of the word in a sentence. We used `lemmatize_strings` function from the `textstem` library on our 'headline_text' column to achieve lemmatisation. The function takes a vector as input and you can also pass lemma dictionary. By default, it uses the `hash_lemmas_dictionary`. (Rinker & Benoit, 2017) (Měchura, n.d.)

1.2.4 Tokenize

Tokenization allows us to split a text in small chunks which creates vectors of our inputs. These vectors can be perceived by feature learning algorithms on different layers which in turn will feed our ML algorithm. Tokenisation in English is a relatively trivial problem as one can tokenize based on white space but for other languages it can provide quite difficult. (Prakrankamanant & Chuangsuwanich, 2022). We used the `step_tokenize` function chained with `recipe` under the `tidymodels` package. We limited the number of tokens to 500 in the entire corpus and keep only the most frequently occurring 500 tokens. This will save us a lot of process time use a lot less memory.

1.2.4 Down sample

Imbalanced class problem has been a thorn in researchers' steps for quite a while. In recent times many papers have been published providing multiple methods to try to manage the problem. (Haixiang, et al., 2017) Down sampling or Under sampling is a way to reduce observations in the majority class in a way that it lines up with the minority classes. There are different methods one can approach while down sampling, however, in the paper we have used random under sampling where observations from the larger class are randomly selected and removed to bring down the size. `step_downsample` in the `tidymodels` package was used to achieve the result.

1.3 Feature selection

Document representation allows us to extract or select features as a numerical vector from a corpus of text. There are many methods of document representation available, TF-IDF is the one which is widely applied (Beel, Gipp, Langer, & Breitinger, 2016) and has the longest history of application. (Robertson, 2004)

Term frequency – inverse document frequency (TF-IDF)

TF-IDF is based on 2 basic concepts.

1. A document is a collection of words (bag of words scheme) - TF
2. If a word is important to a document, it appears more in that document and less in others - IDF

TF-IDF equation is defined as below.

Equation 1: TF-IDF

$$TF - IDF_{ij} = tf_{ij} * \log\left(\frac{N}{df_i + 1}\right)$$

Term frequency *tf* of a word *i* in a document *j* is the number of times the word appears in that document. Document frequency *df* is the number of documents *j* where that word *i* appears.

The larger the value of *tf* the more important the word is, and the larger the value of *df*, the more common the word is. Thus, an important for document, would have a large *tf* and a small *df*. This way common words get a smaller number and have much lesser weight when classifying a document. (Kim, Seo, Cho, & Kang, 2019) In our paper we used the `step_tfidf` function in `tidymodels` package to get the features vectors.

2. METHODS

2.1 Modelling

We prepared the dataset and split it in a `train_set` and `test_set` with an 80% split with stratified random sampling based on *category.* We used the `initial_split` function to achieve the split.

```
data_split    = initial_split(data = all_data, strata = category, prop = .8)

train_set     = training(data_split)
test_set      = testing(data_split)
```

Figure 5: Code snippet: train - test split

We then created a `recipe` to apply pre-processing functions to our set before running it through the models. Model is set by calling a model function with parameters, chaining the relevant mode and engine.

```
train_rec     = recipe(category ~ headline_text, data = train_set)
```

```
train_rec     = train_rec %>%
  step_tokenize(headline_text) %>%
  step_tokenfilter(headline_text, max_tokens = 500) %>%
  step_downsample(category) %>%
  step_tfidf(headline_text)
```

Figure 6: Code snippet: tidymodels recipe

Then Model training pipeline is passed as a `workflow()` and then fitted to the training data. We then defined functions to predict results on the test data, compare it to actual results and collect the metrics and time taken for each model to train such predict.

```
null_spec       = null_model() %>%
  set_mode("classification") %>%
  set_engine("parsnip")
```

```
# Build pipeline
null_wf         = workflow() %>%
  add_recipe(train_rec) %>%
  add_model(null_spec)
# Fit the model
null_fit        = fit(null_wf,data = train_set)
```

Figure 7: Code snippet: Example of model fit

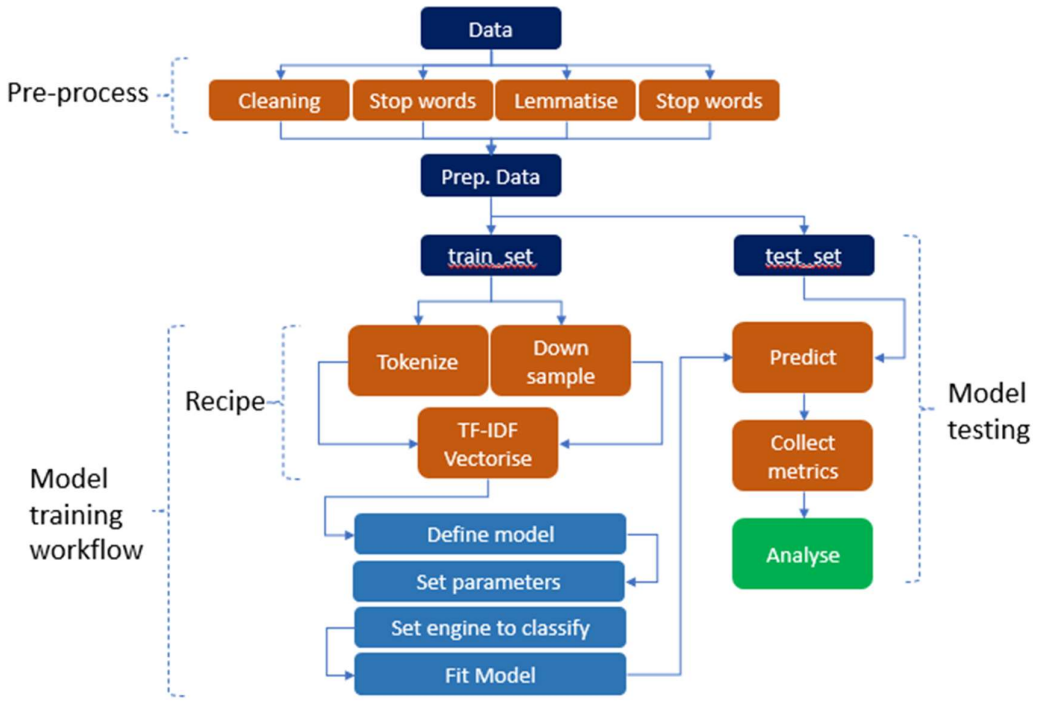

Figure 8:Classification workflow

2.1.1 Naïve Bayes

Naïve bayes analyses each feature independently and calculates the probability of being classified to a particular label. Naïve bayes is not a single algorithm but a collection of many algorithms and

works on each feature independently. (Ying, Mursitama, Shidarta, & Lohansen, 2021) Thus, it is not affected by inter feature relationships and is easy to implement. Naïve bayes in R `tidymodels` can be called as a function `naive_Bayes()`

```
nb_spec        = naive_Bayes() %>%
  set_mode("classification") %>%
  set_engine("naivebayes")
```

Figure 9: Code snippet: Naive bayes implementation

2.1.2 KNN

K-nearest neighbours is a non-parameterized way of classification. It is a rather simple algorithm that classifies the `k`, which is an input, nearest neighbouring points to be part of the same group classified in the same category. (Chen, Zhou, Li, Zhang, & Huo, 2020) It has two main parameters, *distance*, and *k*. We left the distance metric to be `optimal` and started with the default `k` of `5`. It is a rather easy on to implement and is widely used in text classification problems. (Yong, 2009) However, for larger datasets with multiple classes it might take an abnormally large amount of time to compute. KNN in R `tidymodels` can be called as a function `nearest_neighbour()`

```
knn_spec        = nearest_neighbor(neighbors = 5, weight_func = "optimal") %>%
                  set_mode("classification") %>%
                  set_engine("kknn")
```

Figure 10: Code snippet: KNN implementation

2.1.3 Decision trees

Decision tree is a non-parameterized classifier for text categorization represented in form of a tree in which each node can act as leaf or decision node. (Kaur & Bajaj, 2016) Decision trees employ a greedy search algorithm and identifies optimal split points to arrive at node which classifies the observation to a label. Decision trees can be highly effective if the trees are smaller, i.e., the search can be completed in a few steps to identify the class, however this will depend on the nature of the dataset. As trees grow larger, we run into the risk of overfitting. Decision trees can very effective and can be expanded to be used in Ensemble methods

```
tree_spec       =  decision_tree() %>%
                     set_mode("classification") %>%
                     set_engine("C5.0")
```

Figure 11: Code snippet: Decision tree implementation

2.1.4 Random Forest

Rather than using one decision tree, random forest allows us to build multiple decision trees, through *bagging* technique, and combines them together, allowing us to create a more accurate predictive model. Random subset of features is used to build each tree and results are averaged. We used the `rand_forest` function in the `tidymodels` library in R which provides the ability to limit the total number or trees it is allowed to create to arrive at the nodes. The time taken to process will vary depending on the nature of the data and parameters set by the user. We limit the number of trees to 200 and minimum nodes to 5.

```
rf_cls_spec     = rand_forest(trees = 200, min_n = 5) %>%
                    # This model can be used for classification or regression
                    set_mode("classification") %>%
                    set_engine("randomForest")
```

Figure 12: Code snippet: Random Forest implementation

2.1.5 XG Boost trees

Like Random Forest, XG (eXtreme Gradient) boost trees is also an ensemble method where it creates multiple decision trees to learn and classify. The difference, however, is how the trees are created. When a new tree is added the parameters of the new one is adjusted to minimize the loss thus as more trees are added more loss is minimised. It is based on the idea of boosting, hence the term gradient boosting where one single weak learner (tree) is combined with multiple weak leaners (trees) and in turn creates a strong model. (NVIDIA, 2022). For XG boost we also limit the no. of trees to 200.

```
xgboost_cls_spec     = boost_tree(trees = 200) %>%
                       set_mode("classification") %>%
                       set_engine("xgboost")
```

Figure 13: Code snippet: XG Boosted trees implementation

2.1.6 Support vector machines

Classification with SVM involves finding an optimal hyperplane that separates two classes of points, called positive and negative examples, with a substantial margin. (Sun, Lim, & Liu, 2009) As the hyperplane learned by SVM is defined by support vectors only, it is expected that SVM is less affected by imbalanced training examples (Visa & Ralescu, 2005) In this paper we will implement learning on SVM with linear and polynomial modes. We can call `svm_linear()` in `tidymodels` R to apply a linear SVM function and `svm_poly()` to call the polynomial function.

```
svm_cls_spec     = svm_linear(cost = 1) %>%
                     # This model can be used for classification
                     set_mode("classification") %>%
                     set_engine("LiblineaR")

svm_poly_cls_spec = svm_poly(cost = 1) %>%
                      # This model can be used for classification
                      set_mode("classification") %>%
                      set_engine("kernlab")
```

Figure 14: Code snippet: Support vector model implementation

2.1.7 Logistic regression

Logistic regression uses probabilistic methods to classify in a binary category. One can also use multinomial logistic regression to train on multiple input variables and can predict more than two possible discreet outcomes. In our paper, we used the KERAS engine for computation. Keras is an open-source Python library which is designed to create rapid models with deep neural networks and thus is recommended for rapid and simple prototyping. (Benbrahim, Hachimi, & Amine, 2021)

```
mr_kr_cls_spec     = multinom_reg(penalty = 0.1) %>%
                       set_mode("classification") %>%
                       set_engine("keras")
```

Figure 15: Code snippet: Multinomial LR implementation

2.2 Model evaluation

After applying each modelling technique, we listed above, we predict the results of the `test_set`. Then we compare the estimated label by the model to the actual labels evaluate the results based on the following metrics. (Xu, Zhang, & Miao, 2020)

2.2.1 Confusion matrix

Confusion matrix is a way of assessing performance of a supervised learning model by representing the outcome in a tabular format, where columns are the predicted/estimated labels and rows are the true/actual labels. (Xu, Zhang, & Miao, 2020) The matrix defines quadrants as True positive (TP), True negative (TN), False positive (FP), and False negatives (FN). While for binary classification visualisation of a confusion matrix is simple, for multiclass it may become difficult to plot and visualise depending on the number of labels. The variables we derive in the confidence allows us to create metrics we provide below. `Tidymodels` R provide a `conf_mat()` function which takes `truth` and `estimate` as input and provide a confusion matrix as a dataframe.

2.2.2 Accuracy

Accuracy is simply defined as the no of predictions that model was able to correctly predict. We can define accuracy as an equation.

Equation 2: Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

While accuracy is a widely used evaluation metric for text classification models, for imbalanced classes it is known to be non-reliable. In an imbalanced class any model can just assign the prediction to the majority class and get a better accuracy because the sample is mostly consisted of such majority. (Chen, Chen, Hsu, & Zeng, 2008) Thus, for this paper we'll also look at some other metrics when evaluating the model. `accuracy()` function allows us to fetch the accuracy of the predictions.

2.2.3 Precision

Precision describes the portion of actually correct observations from the positive predictions that the model made. For example, a precision of 30% means that the model was able to correctly predict a positive class, 30% of the time. We can define it under an equation as,

Equation 3: Precision

$$Precision = \frac{TP}{TP + FP}$$

`precision()` function in R allows us to fetch the accuracy of the predictions.

2.2.4 Recall

Recall describes the portion of actually correct predictions that were correctly identified. For example, a recall of 70% says that the model was able to predict 70% of the total positive class. We can define recall's equation as,

Equation 4: Recall

$$Recall = \frac{TP}{TP + FN}$$

`recall()` function allows us to fetch the accuracy of the predictions.

2.2.5 F1 score

Precision and recall are two very important evaluation metrics, and they should be taken into account together rather than in solitary when measuring model performance. Usually, tunning the model for better precision results in decreasing recall. Thus, depending on the domain needs, preference should be adjusted. One can combine both those metrics by calculating their harmonic mean, commonly known as F1 score. (Yang, Miller, Jiang, & Moghtaderi, 2020) It can be defined in an equation as,

Equation 5: F1 Score

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

2.2.6 ROC-AUC

ROC is a probability curve that is plotted with False positive rate plotted on the x-axis and True positive rate on the y-axis. ROC (Receiver operating characteristic curve) shows performance of the model at different classification thresholds. AUC measure the two-dimensional area under the ROC curve which represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting. (Narkhede, 2018)

Equation 6: True positive and False positive rate

$$True\ Positive\ rate\ (TPR) = Recall = \frac{TP}{TP + FN}$$

$$False\ Positive\ rate\ (TPR) = \frac{FP}{FP + TN}$$

`roc_aunp()` function allows us to fetch the accuracy of the predictions.

2.2.7 Process time

The last measure we'll assess is process time. Time to process any algorithm is a valuable information to have especially when you are trying to optimize performance with limited resources. Moreover, it comes in handy when deciding a tie between two models presenting the similar performance on other metrics. We'll determine the time by measuring the difference between system time (`Sys.time()` allows us to get system time in R) at the start and end of the train workflow. We'll present these measures in seconds.

3. RESULTS

To compare model performance, we designed a function in R that predicts the labels for each model on the `test_set` and gets the set metrics for each model in a vector.

```r
# Creating a function to collect metrics
model_metrics = function(model_fit, test_set, label_col, model_name)
{
  class_predictions = predict(model_fit, test_set)
  prob_predictions  = tryCatch({
    predict(model_fit, test_set, type = "prob")
  },
  error=function(cond){
    return (NULL)
  })
  met_accuracy      = bind_cols(test_set,class_predictions) %>%
    accuracy(truth = label_col, estimate = .pred_class)
  met_precision     = bind_cols(test_set,class_predictions) %>%
    precision(truth = label_col, estimate = .pred_class)
  met_recall        = bind_cols(test_set,class_predictions) %>%
    recall(truth = label_col, estimate = .pred_class)

  met_roc           = tryCatch({
    bind_cols(test_set,prob_predictions) %>%
      roc_aunp(label_col, names(prob_predictions)) %>%
      mutate(Model=model_name)
  },
  error=function(cond){
    return (data.frame(.estimate = c(0)))
  })
  return (c(model_name,met_accuracy$.estimate,
                       met_precision$.estimate,
                       met_recall$.estimate,
                       met_roc$.estimate))
}
```

Figure 16: Code snippet: Function to collect metrics

Once all models have done predicting results, we collect all metric vectors in a data frame. We then calculate the Equation 5 provided above taking Precision and Recall for all models and calculate the F1 score for all models. You can observe metrics for all models in Table 1. (Please note that the run times might differ based on the machine you are using to recreate the analysis)

Table 1: Model performance comparison

Model	Accuracy	Precision	Recall	ROC AUC	F1Score	Process Time seconds
Null model	0.0142	0.0142	0.0384	0.5000	0.0207	2.19
Naive Bayes	0.2921	0.0525	0.0341	0.4537	0.0414	20.85
K Nearest Neighbour	0.1492	0.1303	0.1413	0.6147	0.1356	20.88
Decision Tree	0.2021	0.1815	0.2421	0.6543	0.2075	25.82
Random Forest	0.2890	0.2161	0.2829	0.7547	0.2450	59.23
XG Boost	0.2364	0.1901	0.2479	0.7324	0.2152	243.33
SVM Linear	0.2246	0.1788	0.2441	-	0.2064	7.80
SVM Polynomial	0.1998	0.1660	0.2119	0.6218	0.1861	248.56
Multinominal Logistic Regression with KERAS	0.0378	0.0378	0.0385	0.5183	0.03811	24.50

We can observe that no model is very highly accurate at predicting news categories based on provided headline words and most of them have accuracies above the Null model's measured accuracy. The maximum accuracy we observe is 29.21% which is achieved by Naïve Bayes. The decision tree models also provide comparable accuracy lead by Random Forest providing an accuracy of 28.90%, XG boosted trees providing 23.64% and normal Decision tree model at 20.21%. The SVM models provide an accuracy of 22.46% for linear and 19.98% for polynomial. KNN performs the second last at 14.92% and followed by Multinomial Logistic regression measuring only 3.78%.

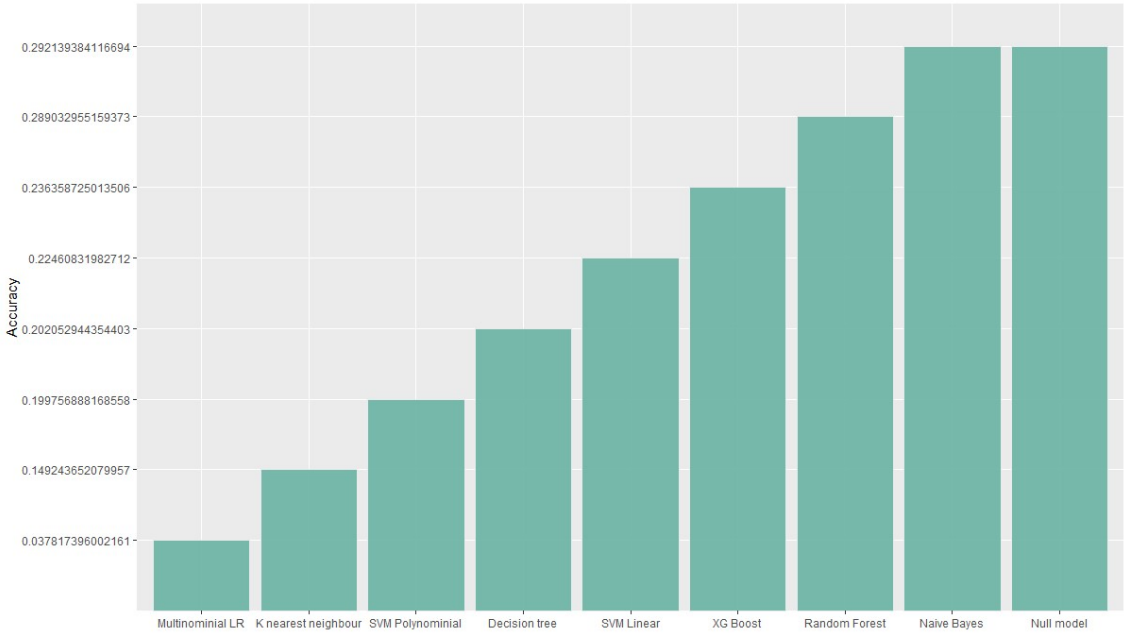

Figure 17: Accuracy plot for all models

However, the nature of our dataset is peculiar as it is an imbalanced class dataset. Thus, just relying on accuracy is not enough. Precision and Recall provide a much better indication on the ability of the model to correctly predict labels. F1 Score combines the ability of both Precision and Recall and provides a single measure. Contrary to accuracy, Naïve bayes model both have an F1 Score of 4.14% providing that it is assigning most observations to the biggest class and thus is not good at predicting News category. Multinomial LR also lags at 3.81% F1 score, while KNN provides 13.56% F1 score. The SVM linear and polynomial models provide F1 scores not far from their accuracy scores at 20.64% and 18.61%. The tree models lead the pack with normal Decision tree coming at 20.75%, XG Boosted trees at 21.52% and Random Forest leading at 24.5%.

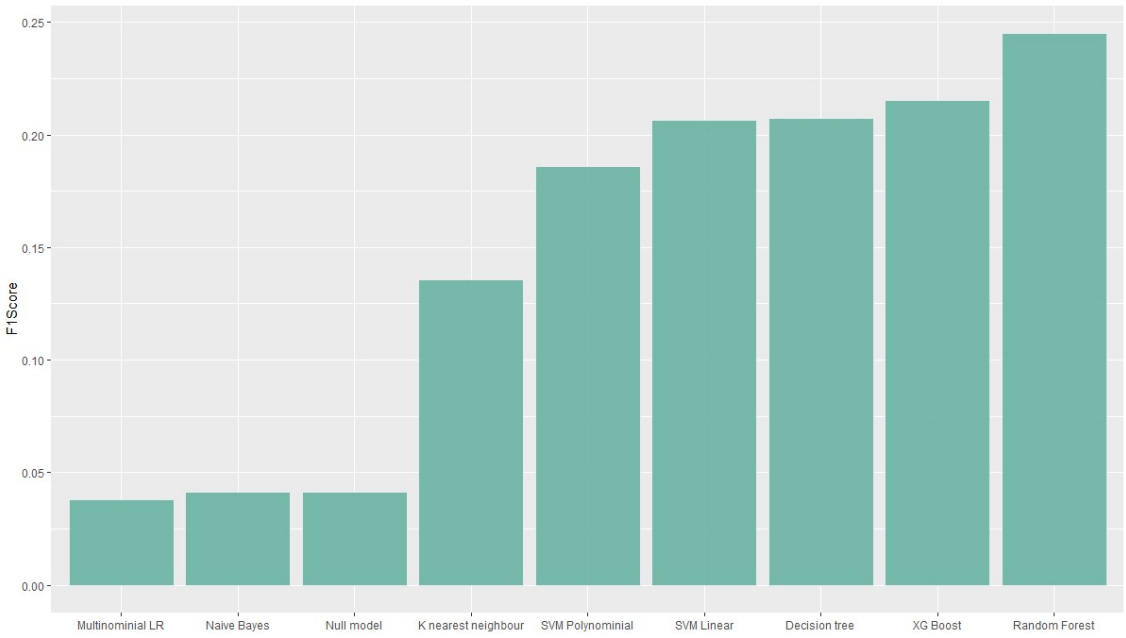

Figure 18: F1 Score plot for all models

4. CONCLUSION

From the above we can see that even if most models are not highly accurate at predicting the category based on the headline words provided, Random Forest and XG Boosted trees do a relatively good job at prediction. They have one of the highest accuracies and also high F1 scores. When we look at the area under curve for ROC we get 0.7324 for XG boosted trees and 0.7547 for Random forest model, both again proving competitive. However, when we look at the process times XG Boosted tree model takes 243.33 seconds to execute and predict, while Random Forest on the other hand only takes 59.23 seconds to execute and predict. Thus, even if most of the models performed to an aspired level, Random Forest seems to show the most promise.

The data set we set out to learn on is a short text dataset our Null hypothesis was that there is enough information in the headline word count to enable a text classification model to predict news categories at or above the level of Null model where level of performance is based on Accuracy and F1 Score. We start with a set of headlines where the number of words fairly normally distributed within a range of 1 to 27, with a median of 11, mean of 10.89 and a SD of 2.77. After text cleaning the range reduces to a count between 4 to 19 with a median of 8, a mean of 8.09 and a SD of 2.002. Thus, working in less amount of information is already a contributing factor to low prediction rates. Our data set also came with the issue of imbalanced class which we tried to mitigate with down sampling. In information filtering short text classification is an evolving field and is a difficult task. (Sriram, Fuhry, Demir, Ferhatosmanoglu, & Demirbas, 2010) . We did manage to prove on all metrics that the Random Forest performs much better than the Null model. However, we did not manage to find a model that can perform at high accuracy and F1 score levels. Thus, we succeed to disprove the Null hypothesis, but much is left to be desired.

5. DISCUSSION

In our process to find the most suitable model we did manage to gather valuable information. We'll discuss a few points where we believe that our research might fall short and further research may work towards finding ways to improve model performance.

We tried to predict based solely on headline words, we did not account for other sets of information like, date of publish, author, short description and others. Many researchers have contributed that adding other information like meta data and ancillary text attributes might be able to improve model performance. (Banerjee, Ramanathan, & Gupta, 2007) For e.g., adding authors name might help predicting labels better as traditionally most authors stick to writing the same category of articles over time.

The paper only explores training and testing based on one-fold of a split of the data set. We did not explore cross-validation and its benefits due to increased run time it entails. Cross validation is a resampling method that portions the data in multiple ways and trains and test each of those folds or iterations. Cross validation allows training to be more confident on prediction of the data as we run tit through multiple sets and reduces the risk of overfitting of the model on a training set. (Jia, 2017)

Hyper parameters of a model are an essential part of model engineering. Depending on the model the nature and number of hypermeters differ. Many models like SVM can do a grid search and find and select the best parameters based on a metric. Tuning hyper parameters can be a resource intensive task and can add substantial process time. (Li, et al., 2022)

6. BIBLIOGRAPHY

Azizi, H., & Reza, H. (2021). Data mining based investigation of the impact of imbalanced dataset over fractured zone detection. *International Journal of Engineering & Technology*(10 (2) (2021)), 124-133.

Banerjee, S., Ramanathan, K., & Gupta, A. (2007, July). Clustering Short Texts using Wikipedia. *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 787-788. doi:https://doi.org/10.1145/1277741.1277909

Beel, J., Gipp, B., Langer, S., & Breitinger, C. (2016). Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries volume*(17), 305-338. doi:10.1007/s00799-015-0156-0

Benbrahim, H., Hachimi, H., & Amine, A. (2021, June). Deep Transfer Learning Pipelines with Apache Spark and Keras TensorFlow combined with Logistic Regression to Detect COVID-19 in Chest CT Images. *WALAILAK JOURNAL OF SCIENCE AND TECHNOLOGY, 18*(11), -. doi: https://doi.org/10.48048/wjst.2021.13109

Chang, R.-S., & Chang, H.-P. (2008, January 26). A dynamic data replication strategy using access-weights in data grids. *J Supercomput (2008)*(45), 277–295. doi:DOI 10.1007/s11227-008-0172-6

Chen, M.-C., Chen, L.-S., Hsu, C.-C., & Zeng, W.-R. (2008). An information granulation based data mining approach for classifying imbalanced data. *Information Sciences*(178), 3214-3227.

Chen, Z., Zhou, L. J., Li, X. D., Zhang, J. N., & Huo, W. J. (2020). The Lao Text Classification Method Based on KNN. *Procedia Computer Science*(166), 523-528. doi:10.1016/j.procs.2020.02.053

Dai, Z., Li, K., Li, H., & Li, X. (2020). An Unsupervised Learning Short Text Clustering Method. *Journal of Physics: Conference Series - 2020 International Conference on Applied Physics and Computing (ICAPC 2020)*(1650). doi:doi:10.1088/1742-6596/1650/3/032090

Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., & Binge, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 220-239.

Harald Piringer, Visplore. (2020, November 18). *Data Exploration – What, Why, and How*. Retrieved from Visplore: https://visplore.com/benefits-of-data-exploration/

Jia, Z. (2017, October 20). Controlling the Overfitting of Heritability in Genomic Selection through Cross Validation. *Scientific Reports*, 13678. doi:https://doi.org/10.1038/s41598-017-14070-z

Kaur, G., & Bajaj, K. (2016). News Classification and Its Techniques: A Review. *IOSR Journal of Computer Engineering (IOSR-JCE), 18*(1), 22-26.

Kim, D., Seo, D., Cho, S., & Kang, P. (2019, March). Multi-co-training for document classification using various document representations: TF–IDF, LDA, and Doc2Vec. *Information Sciences, 477*, 15-29.

Li, Y., Shen, Y., Jiang, H., Zhang, W., Li, J., Liu, J., . . . Cui, B. (2022, February). Hyper-tune: towards efficient hyper-parameter tuning at scale. *Proceedings of the VLDB Endowment, 15*(6), 1256-1265. doi:https://doi.org/10.14778/3514061.3514071

Měchura, M. (n.d.). Retrieved from https://www.lexiconista.com/

Misra, R. (2018). *News Category Dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/rmisra/news-category-dataset

Narkhede, S. (2018, June 26). *Understanding AUC - ROC Curve*. Retrieved from Towards data science: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

NVIDIA. (2022, -). *XGBoost*. Retrieved from What is XGBoost?: https://www.nvidia.com/en-us/glossary/data-science/xgboost/#:~:text=XGBoost%2C%20which%20stands%20for%20Extreme,%2C%20classification%2C%20and%20ranking%20problems.

Prakrankamanant, P., & Chuangsuwanich, E. (2022). Tokenization-based data augmentation for text classification. *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, -. doi:10.1109/JCSSE54890.2022.9836268

Rinker, T., & Benoit, K. (2017). *textstem*. Retrieved from textstem GitHub: https://github.com/trinker/textstem

Robertson, S. (2004). Understanding inverse documant frequency: on theoretical arguments for IDF. *Journal of Documentation; Bradford, 60*(5), 503-520.

Shargabi, B. A., Olayah, F., & Romimah, W. A. (2011). An experimental study for the effect of stop words elimination for Arabic text classification algorithms. *International Journal of*

Information Technology and Web Engineering, 6(2), -. doi:http://dx.doi.org/10.4018/jitwe.2011040106

Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., & Demirbas, M. (2010). Short Text Classification in Twitter to Improve Information Filtering. *SIGIR '10: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 841-842. doi:https://doi.org/10.1145/1835449.1835643

Sun, A., Lim, E.-P., & Liu, Y. (2009). On strategies for imbalanced text classification using SVM: A comparative study. *Decision Support Systems*(48), 191-201. doi:10.1016/j.dss.2009.07.011

The Huffington Post. (n.d.). Retrieved from https://www.huffpost.com/

Tidymodels. (n.d.). *What is tidy models*. Retrieved from Tidymodels: https://www.tidymodels.org/

Visa, S., & Ralescu, A. (2005). Issues in Mining Imbalanced Data Sets- A Review Paper. *Proceedings of the sixteen midwest artificial intelligence and cognitive science conference, 2005*, 67-73.

Wikipedia. (2022). *Confusion matrix*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Confusion_matrix

Xu, J., Zhang, Y., & Miao, D. (2020). Three-way confusion matrix for classification: A measure driven view. *Information Sciences, 507*, 772-794.

Yang, Y., Miller, C., Jiang, P., & Moghtaderi, A. (2020). Case Study of Multi-class Classification with Diversified Precision Recall Requirements for Query Disambiguation. *SIGIR '20: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1633-1636. doi:https://doi.org/10.1145/3397271.3401315

Ying, Y., Mursitama, T. N., Shidarta, & Lohansen. (2021). Effectiveness of the News Text Classification Test Using the Naïve Bayes' Classification Text Mining Method. *Journal of Physics: Conference Series, 1764*, -. doi:10.1088/1742-6596/1764/1/012105

Yong, Z. (2009). An Improved KNN Text Classification Algorithm. *JOURNAL OF COMPUTERS, 4*(3). doi:10.4304/jcp.4.3.230-237