

The Analytics Edge
(Summer 2022)
Data Competition Report
by Team 18

Table of Contents

1. High-level description of the approach developed.....	1
1.1 <i>Restating the problem statement</i>	1
1.2 <i>Our problem-solving strategy</i>	1
1.3 <i>Other details to note</i>	2
2. Description of the results	2
3. Interpretability of the results	3
4. Limitations of the approach.....	3
5. Key takeaways from the TAE Data Competition	4

1. High-level description of the approach developed

1.1 Restating the problem statement:

The problem statement of the TAE Data Competition 2022 requires students to develop an algorithm that discerns and classifies tweets based on their general sentiment. Essentially, the task is to use R programming to perform sentiment analysis on text data and classify tweets as positives, negatives, or neutrals.

1.2 Our problem-solving strategy:

Our strategy was to maximise our twice-daily submissions by validating our test set accuracy using multiple tried and tested classification models. Before building classification models for predicting the sentiments of tweets on the test dataset, we had to pre-process the text data.

- Firstly, using R's text mining library, *tm*, we created a corpus consisting of all 20,610 unprocessed tweets from the train.csv dataset.
- Next, we cleaned the corpus by removing English stopwords and punctuations. The text in the corpus were all made lowercase. We also stemmed the words to reduce the occurrence of texts with the same root words.
- Finally, we created a Document Term Matrix (DTM), removed sparse terms, and converted it into a data frame.

Next, we proceeded to build the following classification models using the training set in order to get tweet sentiment predictions on our test data:

- Random Forests
- Logistic regression
- Support Vector Machine
- CART classifier
- Bagging model
- Naïve Bayes classifier

We also experimented with model creation using external R packages such as *mlr3*, *h2o*, *sentimentr* and *syuzhet* to perform sentiment analysis and classification. Libraries such as *h2o* for instance, uses a more computationally efficient method and hence boasts a shorter model learning period. In addition, they provide higher level functionalities such as model ensembling and training data predictions.

Additionally, our team attempted to create an Artificial Neural Network as a classification model. However, we had to drop the idea since it was computationally intensive and time-consuming.

The top three models – **Random Forests** (*Public score: 0.67467*), **Logistic Regression** (*Public score: 0.67176*), and **SVM** (*Public score: 0.67248*) – were our

best performing models on the Kaggle platform. This report predominantly covers the approaches used in the creation of these three classification models.

(Note: The code for all classification models built is available in the zipped folder)

1.3 Other details to take note of:

- The train.csv dataset has a 29-40-31 split of negative, neutral, and negative tweets. Since unbalanced data could hinder the performance of classification models by introducing biasness towards choosing the more prominent classes (neutral), we had initially decided to down sample the training data in an attempt to fix the class imbalance during our initial modelling. However, we later discovered that down sampling did more harm than good in terms of model performance. And so, we did not carry out this procedure on the train data for the subsequent classification models we built.

```
> #Fixing class imbalance
> three <- which(train$sentiment == 3)
> two <- which(train$sentiment == 2)
> one <- which(train$sentiment == 1)
> three_downsampled <- sample(three, length(one))
> two_downsampled <- sample(two, length(one))
> downsampled <- train[c(three_downsampled, two_downsampled, one),]
> table(downsampled$sentiment)
```

	1	2	3
	5836	5836	5836

2. Description of results

We chose the predictions generated by Random Forests as our final submission since it had the highest public score of 0.67467 on the Kaggle leaderboard.

```
> table(predictforest) Our Random Forests model predicted 1,301 negative
predictforest          sentiments, 3,687 neutral sentiments and 1,883 positive
  1     2     3
1301 3687 1883
sentiments in the test data.
```

Since Random Forests are inherently capable of performing multiclass classification as opposed to binary classifiers such as Logistic Regression and SVM, the prediction results were relatively easier to interpret. This comes across as counter-intuitive because theoretically, the results of Random Forests are harder to interpret than those of the Logistic Regression model. But since this was a multiclass classification problem, it is more logical to choose Random Forest since more steps had to be performed along with Logistic Regression to get the final predictions.

When using Random Forests to make predictions on test data, there was no need to perform One-vs-Rest heuristic method to compare the predicted probabilities of all classes (positive, negative, and neutral sentiments) and then choose the class with the highest prediction probability value as the prediction for every tweet in the test data like we did for Logistic Regression and SVM.

```

> logreg_predictions <- 1:6871
> logreg_predictions <- as.data.frame(logreg_predictions)
> logreg_predictions$Pos_Prob <- Pos_predict$Pos_predict
> logreg_predictions$Neu_Prob <- Neu_predict$Neu_predict
> logreg_predictions$Neg_Prob <- Neg_predict$Neg_predict
> logreg_predictions$sentiment_3 <- ifelse(logreg_predictions$Pos_Prob > logreg_predictions$Neg_Prob
& logreg_predictions$Pos_Prob > logreg_predictions$Neu_Prob, 3, 0)
> logreg_predictions$sentiment_2 <- ifelse(logreg_predictions$Neu_Prob > logreg_predictions$Neg_Prob
& logreg_predictions$Neu_Prob > logreg_predictions$Pos_Prob, 2, 0)
> logreg_predictions$sentiment_1 <- ifelse(logreg_predictions$Neg_Prob > logreg_predictions$Neu_Prob
& logreg_predictions$Neg_Prob > logreg_predictions$Pos_Prob, 1, 0)
> logreg_predictions$sentiment <- logreg_predictions$sentiment_3 + logreg_predictions$sentiment_2 +
logreg_predictions$sentiment_1
> head(logreg_predictions)
  logreg_predictions  Pos_Prob  Neu_Prob  Neg_Prob sentiment_3 sentiment_2 sentiment_1 sentiment
1                1  0.3068057  0.4430623  0.20262713          0          2          0          2
2                2  0.6181739  0.1829216  0.03510680          3          0          0          3
3                3  0.2016542  0.5769819  0.19877193          0          2          0          2
4                4  0.1921896  0.5266581  0.26485594          0          2          0          2
5                5  0.5732238  0.3270457  0.09362331          3          0          0          3
6                6  0.1570307  0.5626867  0.26536916          0          2          0          2

```

3. Interpretability of the results

The prediction results of our Random Forests model return values of 1, 2, or 3 for all 6,871 test tweets, with an accuracy of 65.732% in the private leaderboard, and 67.467% in the public leaderboard. It is a respectable degree of accuracy noting that the test tweets were classified based on a model that learns from a limited training set of only 20,610 tweets – meagre compared to the millions of training data used to train industry-winning NLP models such as BERT.

```

> # Random Forests
> set.seed(123)
> library(randomForest)
> RF_model <- randomForest(as.factor(sentiment) ~ ., data = new_train)
> predictforest <- predict(RF_model, newdata = new_test, type="class")
> predictforest #RF predictions
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21
  3   3   2   2   3   2   2   2   2   2   2   1   3   3   2   2   3   2   1   2   1
22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42
  2   2   1   3   3   1   2   1   3   2   3   2   2   3   2   2   2   2   3   2   3
43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  1   3   2   2   2   2   1   3   1   1   2   3   1   2   2   3   1   2   1   2   2
64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84
  3   2   3   1   3   1   3   2   2   2   1   2   2   2   2   3   2   3   3   1   3
85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105

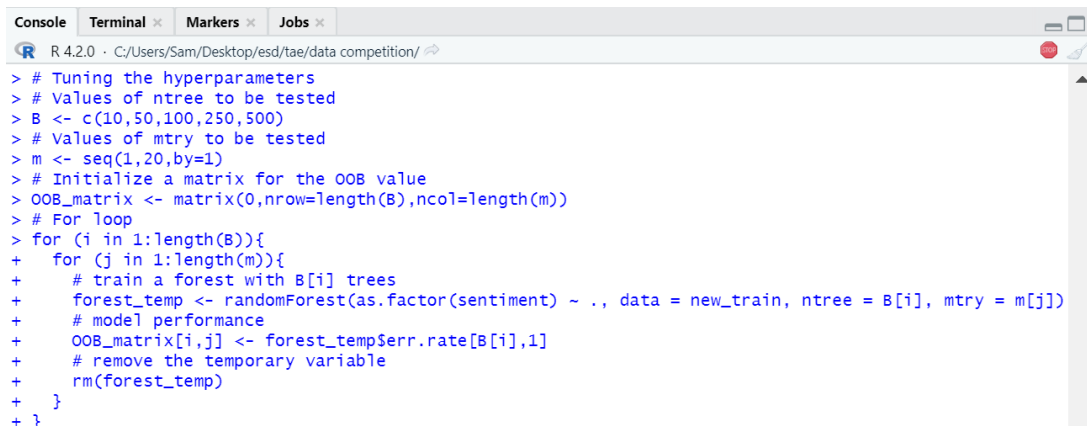
```

4. Limitations of the approach

- In order to increase the accuracy of our Random Forests model, we could have tuned the hyperparameters – mtry (number of randomly chosen predictors used to split a node), ntree (number of trees in the forest) and nodesize (minimum size of terminal leaves). Training our model with hyperparameters providing the best OOB performance could have improved the overall model performance and increased the accuracy score for predictions made on the test data.

Referring to the Week 9 lecture notes, we attempted to create an OOB matrix using for loop that iterates through different combinations of mtry, ntree and nodesize. However, it was extremely time-consuming and computationally

intensive. We were unable to deduce the hyperparameters values for the best-performing Random Forests model with the lowest OOB error.



```
R 4.2.0 · C:/Users/Sam/Desktop/esd/tae/data competition/
> # Tuning the hyperparameters
> # Values of ntree to be tested
> B <- c(10,50,100,250,500)
> # Values of mtry to be tested
> m <- seq(1,20,by=1)
> # Initialize a matrix for the OOB value
> OOB_matrix <- matrix(0,nrow=length(B),ncol=length(m))
> # For loop
> for (i in 1:length(B)){
+   for (j in 1:length(m)){
+     # train a forest with B[i] trees
+     forest_temp <- randomForest(as.factor(sentiment) ~ ., data = new_train, ntree = B[i], mtry = m[j])
+     # model performance
+     OOB_matrix[i,j] <- forest_temp$err.rate[B[i],1]
+     # remove the temporary variable
+     rm(forest_temp)
+   }
+ }
```

- In Logistic Regression, we could have looked up for ways to automate the One-vs-Rest strategy in R instead of manually splitting train data based on three classes and getting the prediction probabilities of the test tweets separately.
- We noticed a decrease in performance of the logistic regression model after training the model on downsampled train data (*Public score: 0.66472*). The accuracy score of the model's predictions was higher when trained using data that was not downsampled. This could be because the downsampling process which was intended to fix class imbalance was only performed on the majority classes of the train data which could have adversely affected the performance of our classification models. In retrospect, a different method to fix class imbalance could have been implemented.
- We built a Naïve Bayes classifier – a multiclass classification model often used in Natural Language Processing – to perform sentiment analysis. We chose this model as it usually has very good model performance in real-world applications. However, it only holds true when the predictors are independent. It performed very poorly on the test data, yielding a public score of 0.57860.

5. Key takeaways from the TAE Data Competition

The Data Competition was a great way to apply machine learning algorithms learnt in the course to solve an NLP problem. It also enabled us to open our minds to search and self-learn other classification models not taught in class. Upon applying various classification models, we realised how similar the performance of these classifiers was like. It is often more fruitful to perform ensemble modelling, test performance of several models on a validation data set, pick the best classifier and look for ways to improve it rather than focusing on separately implementing more complex models to decrease variance in the predicted test data values. As the famous statistician George E. P. Box once said, “*all models are wrong, some are useful*”.