**40.016 The Analytics Edge**

# TAE Competition Report

## Team 11


**Team Members:**

Chia Yu Ying (1004609)

Suzanne-Kae Rocknathan (1004617)

Vaishnavi Divya Shridar (1004600)

## Overview of Approach

Our approach involved three main steps – pre-processing, modelling, and model evaluation – that were repeated until the final model selection took place.

Initially, we only applied the pre-processing techniques learnt in class. This included the removal of punctuations, numbers, English stopwords and extra whitespaces, stemming, the conversion of all text to lowercase, and the removal of terms with at least 95% sparsity. These initial steps were enough to achieve a relatively high accuracy of ~85% on some of the models that we have tried. However, we noticed that removing all punctuations from the raw data may not be a good idea because a number of data entries contained emoticons, which is a potential indicator of sentiment. Hence, we explored functions to improve on our preprocessing steps, so as to take into account such data features.

## Preprocessing

Our pre-processing involved data cleaning and formatting. The purpose of data cleaning was to standardise the terminology used in every text string, while maintaining any terms that could give insight into its sentiment. The libraries of hunspell and textclean were instrumental in this process. We began by replacing the unicode in each string with their symbol equivalent. As such, '>' was replaced with '>' while '<' was replaced with '<'. This step proved useful when all emoticons were later replaced with words of equivalent meaning. For example, '<3' is used to denote a heart emoticon, however for some of the data entries, it appeared as '<3'. By replacing the unicode before translating the emoticons to word equivalents, we could maintain its meaning. When using the replace_emoticon function, the string 'hello:)' gives an output of 'hello smiley'. However, we found that 'hello :)' would not be changed. To counter the context-sensitive nature of this function, we used both the original function and an edited version of the function, sourced from Stackoverflow[1]. Contractions were then replaced – for example, 'it isn't' was replaced with 'it is not' – so as to ensure consistency in meaning across different text strings. Internet slang was also replaced with their long forms, like 'TGIF' being replaced with 'thank god, it's friday'. We also replaced misspellings with their true forms. Lastly, we replaced word elongations, where unnecessarily repeated characters in a phrase were removed. For example, 'amazinggg' would be replaced with 'amazing'.

The resultant character set from the above text cleaning methods were stored as a cleaned dataset for pre-processing, and then formatted as a corpus. We first removed specific strings that were frequently observed in the dataset. These strings include the unicode for degree fahrenheit, degree celsius, html entity references, and an apostrophe mark. Other terms removed include @mention that was found when tweets were replying to other users, and {link} found when hyperlinks were inserted into a tweet. However, upon comparing the model's performance on the validation set with and without this preprocessing step, we discovered that the model obtained a higher accuracy without this step. Hence these specific strings were not excluded from our final

model. Punctuation and numeric values were removed from the corpus after, so as to not interfere with the unicode removal step. All text was converted to lowercase, with English stop words being removed. Stemming was performed on the resulting corpus using the library SnowballC, before extra white spaces were removed.

The final output was converted to a document term matrix, with only valid character values left. We experimented with the removal of terms with varying levels of sparsity, to determine the optimal level of sparsity removal for each model. For the final model, we chose to remove terms with at least 99.9% sparsity as it was found to have the lowest out-of-bag (OOB) error and highest model accuracy on our validation set when training our random forest *(Refer to paragraph 3 in the results section below)*. We then experimented with removing common terms as well, using a custom function sourced from Stackoverflow[2]. An example of a removed term includes 'weather' – the only term present in more than 50% of the preprocessed tweets. Despite experimenting with this function, our best model was found to have a higher accuracy on the validation set without the removal of common terms, and hence this function was not applied for training our final model.

We applied the same preprocessing steps – with the exception of removing sparse and common terms – to the kaggle test dataset to ensure consistency in data cleaning and formatting between the training and test set.

**Modelling**

A 70%-30% training-validation split, with a seed of 1, was done on the training dataset. This split ratio and seed were kept consistent across all attempted models, so that the validation set would be the same when comparing accuracy across models. Attempted models include Classification Trees, Random Forest, Naive Bayes and Support Vector Classification.

**Model Evaluation**

The accuracy scores from predicting on the validation set were used to evaluate and compare the models, as hyperparameters were tuned and the abovementioned preprocessing steps were experimented with. As we noticed the specific changes that led to greater accuracy improvements, we reiterated through the preprocessing, modelling and evaluation process to identify the best set of preprocessing functions and parameters for the models.

**Model Selection**

After model evaluation, the accuracy scores were used to determine the best performing model across all those attempted as the same validation set was used across different models. The top two highest performing models for each day were selected to run against the test set available on Kaggle. Thus, ten different models were uploaded to Kaggle for testing.

**Results**

As mentioned, the 4 main models attempted were Classification Trees, Random Forest, Naive Bayes and Support Vector Classification, and of the 4, the Random Forest was chosen for our final model. Classification trees were trialed with 10-fold cross validation and the effects of pruning were explored as well, with the pruned tree performing better when tested against the validation set. Bagging was also trialed with 10, 20, 25, 50 and 100 trees.

For Random Forests, the hyperparameters – number of variables used at each split (m) and number of trees (B) – were looped through to identify the highest performing model, by identifying the parameters which gave us the lowest out-of-bag (OOB) error. The values sampled ranged from 10 to 30 for the number of variables (m) , and the instances of 10, 50, 100, 250 and 500 for the number of trees. Of these values, the variable count of 12 with 500 trees gave the lowest OOB score as illustrated by Figure 1 in the Appendix. Hence, our best random forest model was trained using m = 12 and B = 500. We recorded and stored our result in the "OOB_matrix_adjust_tree_m.csv" in our submission folder as this process takes ~4 hours to load.

To investigate how our best Random Forest model would perform with different levels of sparsity removed from our document-term matrix (DTM), we iterated through different sparsity values in the removeSparseTerm function, from 0.980 to 0.999, with an increment of 0.001 sparsity for each loop. For each iteration, we ran through the removeSparseTerm function, and retrained our model with the new pre-processed train dataset. OOB errors of our trained random forest models, together with the accuracies on the validation set were noted down. Figure 2 and 3 in the appendix shows our results for this. From the figures, we can clearly see that as the sparsity level in the removeSparseTerm function increased, the OOB error decreased, thus increasing the accuracy on our validation set. This proves that 0.999 gave us the best model performance out of all the other sparsity values used. Our output results are recorded and stored as "OOB_sparsity_result.csv" and "accuracies_sparsity_result.csv " in our submission folder as this process takes ~10 hours to load.

To tune the Random Forest models, a for loop was created to remove predictor variables with the lowest Gini coefficient. For each loop, the lowest ten predictors were removed, before training occurred again and the next lowest predictor was removed again. This process was repeated for 10 loops, with the OOB error and accuracy on validation set being recorded down. After iterating through the 10 loops, we realized that there was no significant decrease in OOB error or improvement in the validation set accuracy. On top of that, upon manually reviewing the words that were removed from the model, we saw that some of the words, such as "dumb", actually carries negative sentiment. The exclusion of such words could restrict our model's ability to pick up on the sentiments of these terms. Due to the above 2 reasons, we chose our final submission on kaggle to be the model which was trained without the removal of any lowest Gini coefficient.

Other attempted models include the Naive Bayes model and Support Vector Machine for classification (SVC). Our best Naive Bayes model yielded an accuracy of about 77.3% when tested against the validation set. The SVC models were the closest to the Random Forest in terms of accuracy, with results ranging from 82% to 87.3%. Linear SVC models were tuned by adjusting the cost parameter with values C = 1, 0.1, 0.05, 0.01, 0.005. When tested on 30% of the actual test set on Kaggle, the SVC models yielded accuracies of 85.422% (before tuning) and 86.711% (after tuning).

**Discussion & Limitations**

As the preprocessing steps with the textclean library took quite some time to run, we stored the preprocessed data in csv files for easy access due to the need of frequent usage for each team member to train the models. After the final Kaggle submission, when ensuring reproducibility of our final model, we discovered that the same textclean preprocessing code produced datasets that had minor differences for some entries, when run on different team members' laptops. As such, the accuracy of the model on the validation set was not exactly the same when run on different members' systems. We suspect that the issue may lie in our lack of understanding of the textclean and hunspell libraries. It is possible that some of the functions in these libraries used randomization – to correct misspelling or emoticons for example – and as such, it could have been better if we had set seed before using the textclean functions. As we have previously learned, even when setting the seed, R can produce different results on different computer systems and softwares, as such, complete consistency is still not guaranteed. To ensure that the exact same results are obtained when training our final model, it is advisable that the pre-saved train_textclean.csv and test_textclean.csv files in our submission file are loaded into the code just before converting to corpus.

The training of our random forest of m = 12 and tree = 500 was also found to be time-consuming, taking slightly more than an hour to complete. Each iteration of our data, after making changes to our pre-processed data, required our random forest to be re-trained. This repetitive process ended up taking a lot of time. An alternative would have been to train and tune our random forest with a lower number of trees as a tradeoff between the accuracy of our model and the time for completion. As seen in Figure 1 in the Appendix, it can be observed that setting parameters tree = 250 and m = 13 would also obtain a relatively low OOB error. With the number of trees being halved, training a random forest would be more rapid and the remaining time could be used to explore with more types of pre-processed data and a greater range of hyperparameter values.

**References**


1. https://stackoverflow.com/questions/62270337/replace-emoticon-function-incorrectly-replaces-characters-within-a-word-r
2. https://stackoverflow.com/questions/25905144/removing-overly-common-words-occur-in-more-than-80-of-the-documents-in-r


**Appendix**

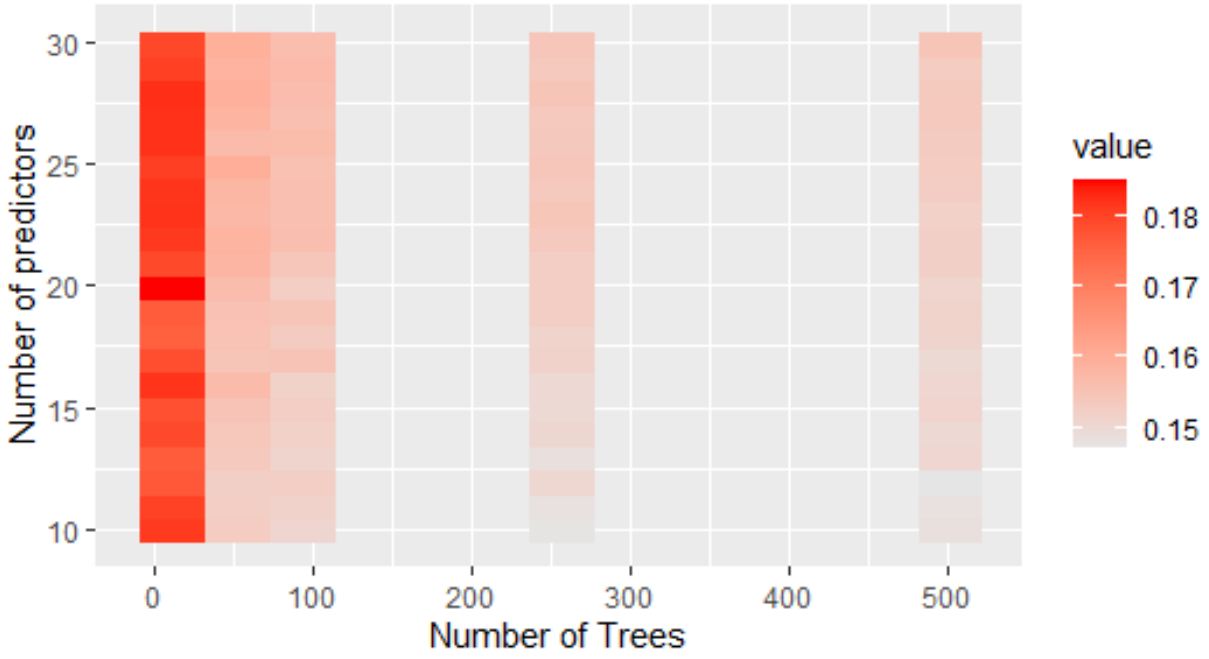


Figure 1: Graph of number subset predictors + number of trees against OOB error

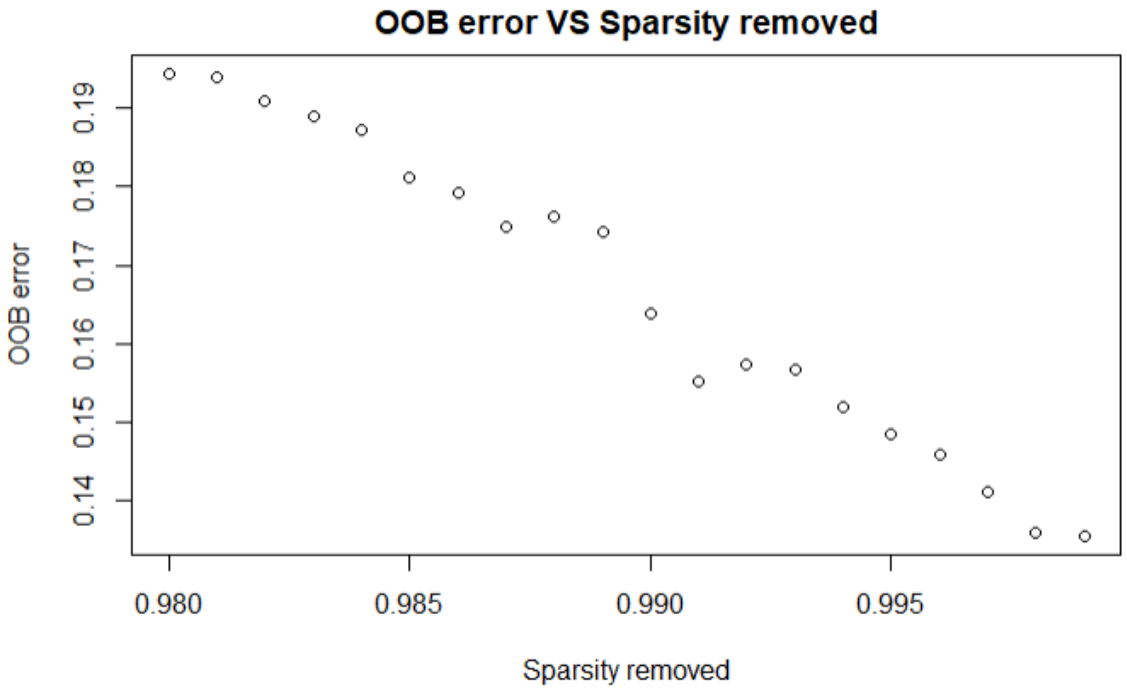


Figure 2: Graph of OOB error against sparsity level removed

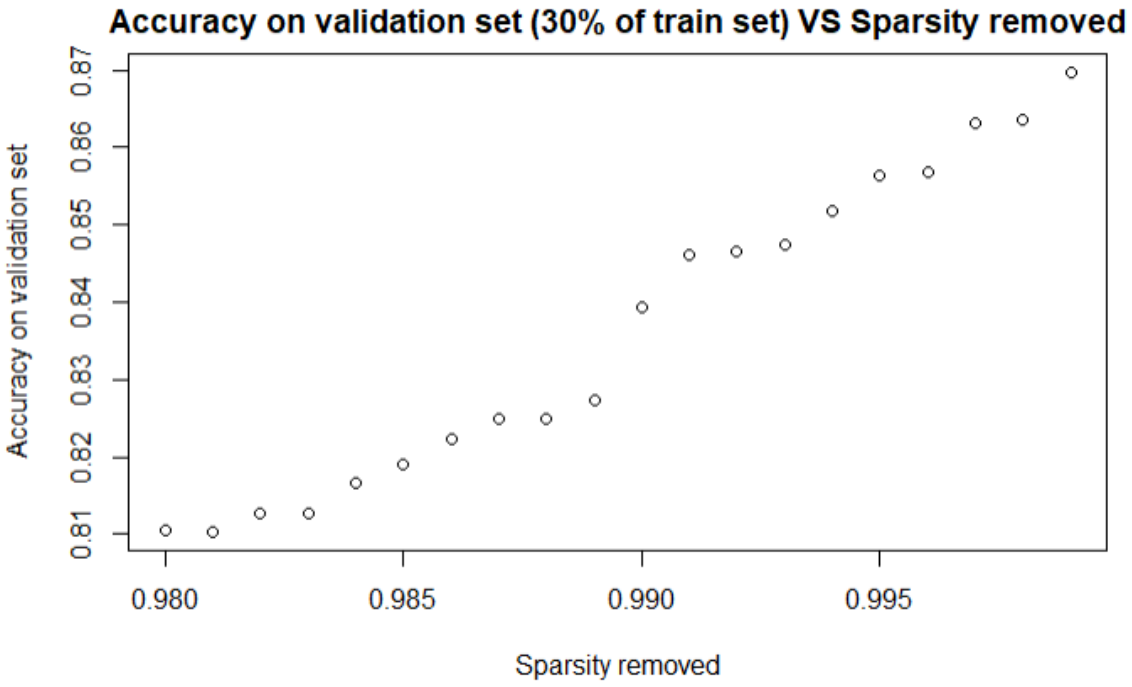


Figure 3: Graph of accuracy on validation set against sparsity level removed