The Analytics Edge (2019) – Data competition

Clarence Toh , Sharan Sunil Pillai , Song Zhiguo

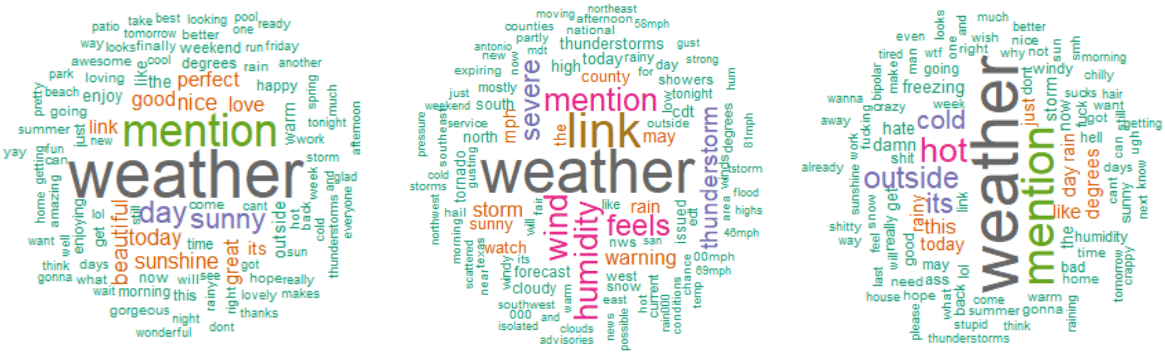
The report will address the team's approach to sentiment classification of tweets relating to the weather. We first begin with data exploration, followed by naive models and then a final approach using Support Vector Machines.

Data Exploration

We first look at the distribution of the sentiment classes across the training data set. We found that the distribution of classes 1, 2 and 3 were balanced. Class 1 made up 31% of the data set, class 2 made up 35% of the data set and class 3 made up 34% of the data set.

1	2	3
0.31	0.35	0.34

We then created word clouds to visualize the most common words in each class. In class 3, beautiful, good, sunny and other words were associated with class 3 (positive sentiment). In class 2, cloudy, humidity, wind, thunderstorm and other words were associated with class 2 (neutral sentiment). In class 1, hot, sucks, cold, outside and other words were associated with class 1 (negative sentiment). (Class 3, 2 and 1 from left to right)



The word clouds tell us what words may be important in this classification process. This will be important as the models will be trained on these words and predict using the corpus consisting of these words. The word clouds also suggest that words like weather and mention may be considered as insignificant words or noise as they are present in all three word clouds.

Data Pre-processing

The data preprocessing includes the conversion of characters from upper case to lower case, removal of punctuation, removal of stopwords and stemming and lemmatization of

the corpus. We created document term matrices of the train set to train our naive models with.

```
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 22500
[1] "ok mom lol rt mention go outsid summer weather"
[1] 22500   910
<<DocumentTermMatrix (documents: 22500, terms: 910)>>
Non-/sparse entries: 141982/20333018
Sparsity           : 99%
Maximal term length: 13
Weighting          : term frequency (tf)
```

```
       Terms
Docs feel hot lol mention mom news old outsid summer weather
   1    0   0   1       1   1    0   0      1      1       1
  10    1   0   0       0   0    0   0      0      0       1
   2    0   1   0       1   0    0   0      1      0       0
   3    0   0   0       1   0    1   1      0      0       0
   4    1   1   0       0   0    0   0      1      0       0
   5    0   0   0       0   0    0   0      0      0       0
   6    0   0   0       1   0    0   0      0      0       0
   7    1   0   0       0   0    0   0      0      0       1
   8    0   0   1       1   0    0   0      0      0       0
   9    0   0   0       1   0    0   0      0      0       1
```

This is a summary of the preprocessing. The document term matrix has a dimensionality of 22500 tweets by 910 unique terms. Printed "ok mom lol rt mention go outsid summer weather" is an example of a pre-processed tweet from the train dataset.

```
 [1] "mention"      "outsid"      "weather"     "hot"         "thunderstorm" "feel"
 [7] "like"         "sunshin"     "link"        "sunni"       "today"        "humid"
[13] "wind"         "get"         "nice"        "day"         "degre"        "love"
[19] "cold"         "rain"        "storm"       "good"        "beauti"       "sever"
```

Above are the terms that appear more than 1000 times in the document term matrix of the corpus.

Models

To build several naive models, we split the training set in a 3:1 ratio, where 75% of the training set will be used to train the model and 25% of the training set will be used for testing. We checked the distribution of the classes across both train and test sets to see if it is balanced. The table below shows that there is enough balance between classes.

```
        1         2         3
0.3100113 0.3560620 0.3339268

        1         2         3
0.3058948 0.3471411 0.3469641
```

CART

The CART model did not perform well, achieving only 0.7453 accuracy. It being a less efficient version of randomForest, we implemented CART as it was a simple model to implement. It was not submitted.

Naive Bayes

Using 10-fold cross validation repeated thrice with laplace smoothing applied, we implemented a Naive Bayes on the training set. The naive Bayes classifier worked out really well, achieving an accuracy of 0.9949. We had suspicions that the model was able to predict very well because the test set was originally part of the corpus that the model was trained on. Hence, there may be a lot of bias. Implementing at the test set, a grid search was added at the cross-validation step and the tuneLength was reduced. This yielded a result of 83% accuracy. This was the first model we submitted and offered a

baseline accuracy for our team to work from. Despite further optimizations attempted, we could not get better results for our Naive Bayes model.

Random forest

Our initial approach was to create a basic 300-tree random forest with no tuning and get a baseline result. The accuracy of the model on the test set was 0.9988. It performed so well that it essentially seemed to be overfitting the data.
To optimize the random forest classifier, tuneRF was utilised to obtain the ideal value for mtry, which is the number of values sampled at each split.

```
#tune for best mtry
bestmtry <- tuneRF(train_set[,-ncol(train_set)], train_set$Class, stepFactor=0.1, improve=1e-5,
ntree=500,trace=T,plot=T,mtryStart = 15)
plot(bestmtry)
##best = 15
```

After minimizing OOB error and selecting mtry to be 15, the model was run for 500 tries on the entire training set.

```
#model
rf_classifier = randomForest(x = train_set[,-ncol(train_set)], y=train_set$Class,data = train_set, ntrees =
500,mtry=15)
rf_pred <- predict(rf_classifier, newdata = test_set[,-ncol(test_set)])
```

This model achieved an accuracy of 83.955% on the public leaderboard, higher than our initial Naive Bayes models.

SVM

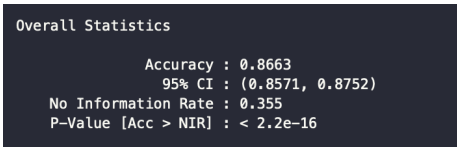
For SVM, the most exploration was conducted. First, we tested each individual kernel type. Clockwise from top-left: Sigmoid, Linear, Polynomial (degree 3), Radial. Observing accuracy and kappa for each model, the linear model performed the best in both. Optimizing the cost for the linear model, we tried various costs and observed the results. The optimal cost for the entire training set was observed to be 0.175.

```
svm_classifier <- svm(Class~., data=train_set,kernel='linear',cost=0.175)
svm_pred <- predict(svm_classifier,test_set[,-ncol(test_set)])
```

This model yielded an accuracy of 84.9% on the public leaderboard and is the final model used for our submission.

XGBoost model for benchmarking

A model was built using XGBoost to benchmark against all our other models. We used mlogloss as our evaluation metric and used 5-fold cross validation to optimize number of rounds at 834. The results on the training set is below. What was more impressive is that it had near identical performance



on the test set. This had an accuracy of 85.8% on the public leaderboard, our highest score and only a drop of 1% from the training set.

Final model

We developed a methodology to identify our optimal model apart from looking solely at the public leaderboard scores alone. The XGBoost model was so far ahead the best performing model. We developed a hypothesis that the closer the SVM/NB/RF model results to our XGboost model, the better its performance. We tested this hypothesis by creating a metric for similarity of each model output to the XGBoost model. The methodology behind it is if the results are the same, return 0, if it differs return 1. Sum up the differences and divide by total rows in the test set to get a percentage of difference between the XGB predictions and the other models. The results are below. The lower the score the better.

	SVM	Random Forest	Naive Bayes
Model Difference	0.05053333	0.0752	0.1078667

We can observe that the SVM differs from the XGB model by 5%, Random Forest by 7.5% and Naive Bayes by 10.8%. Looking at the results from the public leaderboard, the XGB model outperformed SVM, Random Forest and Naive Bayes. This knowledge was used to assume that the XGB Model is State-of-the-art for our group and we chose the model with the lowest difference away from it. Therefore, we chose the SVM with a linear kernel and a cost of 0.175 as our final model for our submission with a final public leaderboard score of 85.822% ranked 16th and a private leaderboard accuracy of 85.466% ranked 18th which means a total test set accuracy of 85.5728%.

Interpretability and limits

Due to the nature of SVM's, the main limitation is that the support vector classifier works by putting data points, above and below the classifying hyper plane there is no probabilistic explanation for the classification. This makes it difficult for us to come up with a clear interpretation of results other than accuracy. It is interesting that a linear kernel performed best on the dataset, which we did not expect. This suggests that the separation between classes is quite distinct and there is little overlap between the various sentiments.

Another limitation of SVM's is if the dataset increases in size, our model would not perform better as it works better with smaller datasets. If the dataset were to increase, we would recommend using our random forest model instead as it would improve with more data added.