HATE SPEECH SENTIMENT EVALUATOR

Sami Chowdhury

AUGUST 4, 2019

Contents

Introduction

The objective of this assignment is to analyze a corpus of tweets and to analyze whether they classify as hate speech. The two sets of features used to perform this analysis were the sentiment plus entropy of each tweet and the term frequency-inverse document frequencies of each tweet. To begin, the sentiments for each tweet were calculated based on the sentiment score values of the meaningful words in the AFINN lexicon. Once these sentiment scores were calculated, a sentiment classification of either "positive", "negative", or "regular" was assigned to each tweet. The next step involved finding the total Shannon entropy for each tweet in the corpus by applying the Shannon entropy equation to each word in every tweet. Once the sum entropy for each tweet was calculated, the two characteristics were grouped into one set of features. The term frequency-inverse document frequencies were then calculated by finding each word's ranking in the corpus as well as its frequency. These two values were multiplied and summed for each tweet and then placed into the second set of testing features. Next, two different support vector machines were built using a polynomial kernel with the features listed above. The speech sentiments were then predicted after the model was trained and tested. Lastly, the model was sent through a prediction table and its evaluation metrics of F1, precision, and recall were calculated. After the cross-validation was complete, the model was then evaluated amongst other support vector machine types to determine the most efficient approach. The results were then graphed and listed to explain the efficiency of the best model.

Theoretical Analysis

Many strategies were implemented in order to perform this specific type of theoretical analysis. In regards to the sentiment analysis, using the AFINN lexicon enables the use of numerical scores to determine sentence sentiment. The issues that arise from this simple method are that it is focused on a unigram style of approach, meaning that word pairs are not taken into account during this type of analysis. For example, if the words "not happy" were to appear together, the method would not be able to properly distinguish that this sentiment is more negative than positive. As such, the numerical scoring method is rudimentary yet still effective. Using a bigram model would have yielded more accurate results.

In terms of the entropy analysis, the methods used were quite simple. The probabilities for each word in each tweet were calculated based on the frequencies they appeared across all tweets. From there, a probability was assigned, and then the Shannon entropy function was applied. The main disadvantage to this method is the operation time, which becomes quite large with the style implemented in the script. Going through the frequency list for each word in each tweet, even with the stop words eliminated, can take up to O(n^2). A simpler method would have been to sum the entropies per tweet when mutating the data frame and then slicing the data frame to produce unique values.

In terms of the term frequency-inverse document frequency analysis, the method used was the most straightforward approach. The term frequencies were calculated similarly to how the frequencies for the entropy section was calculated. The inverse document frequencies were calculated using the bind_tf_idf function in the tidytext package of R. With the current implementation, the main disadvantage was again operation time since all the words across the entire corpus was used. This led to more accurate readings at the cost of lengthier operating times. A faster approach would involve taking a specific large subset of the frequency document so that the most common and significant non-stop words were analyzed.

When finding the best model to classify the tweets, deciding between a support vector machine and a k-Nearest Neighbor classifier method provided differing results. With a support vector machine, radial kernels or classifiers were used in order to classify the tweets. These provided for a higher accuracy and precision when classifying the tweets. The k-Nearest Neighbor classifier algorithm was not as accurate at classifying but proved to be much faster when optimizing the model, as tuning the support vector machine model would take up to 45 minutes depending on the feature set being tested. The tuning for the support vector machine could be enhanced with some of the algorithm adjustments to the entropy and term frequency methods mentioned above. Overall, the support vector machine method was the more useful and appropriate method to use for its much more accurate results than the k-Nearest Neighbor method.

Experimental Setup

This experiment involved a simple yet detailed process to properly perform the analysis. To begin, the corpus of tweets were read into two different data frame variables for separate set analysis. The data frame was then first passed into the sentiment analysis functions. In this section, the sentiment value of each word was calculated based on the numeric sentiment score contained within the AFINN lexicon data frame. These sentiment word score values were summed up for each tweet and then given a sentence sentiment score. Finally, based on the overall score of the tweet, a classification of either positive, negative, or regular was assigned to each tweet.

The next section of the experiment involves the entropy calculations. The tweets were cleaned of their stop words by performing an anti-join between the tweet word set and the stop words dataset. Next, the frequencies for each word in the corpus, grouped by speech and tweet ID, were calculated based on how often they appeared within the entire corpus. After the data was cleaned for any illogical values, the frequencies were kept in a separate data frame. This ensured that the functions created to assign entropy values to tweets would be able to reference it in a piecewise fashion. Similar to the sentiment analysis, an entropy value was calculated based on the probability inputs of each word, and then summed up for the entire tweet. The entropy values were then displayed next to the sentiment decisions to produce the first set of features for our model.

The second set of features involved calculating the term frequency-inverse document frequencies of the words in the corpus. The words in the data frame were piped into a separate data frame tokenized by each word in the text of the tweets. The tokens were then counted by how many tweets they appeared in. The total times each word appeared in the set of tweet IDs for all words was stored into a different data frame so that the relevant information could be grouped together between the words in the tweets and the total words across all tweets. The term frequencies were then calculated for each word in the corpus by dividing the number of times the word appeared by the number of documents the word was observed in. Next, the bind_tf_idf function was applied to the tweet words data frame to calculate the inverse document frequencies of each word. Afterwards, the TF_IDF was calculated by multiplying the term frequency and inverse document frequencies of each word. These TF_IDF's were then summed up for each tweet and sliced so that each tweet ID had one total TF_IDF assigned to it.

The next section involves the classification methods. The two feature sets were created as two data frames: one contained the entropies and sentiments of each tweet while the other contained the TF_IDFs of each tweet. Both data frames also included the speech sentiments provided from the initial input data frame. These sets were then sampled so that 80% of the data was set for training data while 20% of the data was to be used for testing. For the support vector machine method, the models were built to predict the speech sentiments given by the original document based on the training data of each feature set. A standard gamma and cost of 1 was applied for these radial kernel models. The models were then tuned to find the best cost and gamma values to classify the data. A table was then created showing the best cost and gamma values as well as the best performance of each model. A table was then created depicting the predicted and tested values for each set.

The second classification algorithm, k-Nearest Neighbors, was implemented to compare to the support vector machine algorithms. A standard form of the algorithm was applied, focusing on using different k-values to find the best proportion of classification and success rate. The "knn" function in RStudio was applied to find the correct number of classifications applied based on the optimum k-values. The success rates were assessed and the optimal k-value among them was chosen.

The final step was to cross validate the data to measure the F1, precision, and accuracy scores of our classification models. This involved using a 10-fold resampling method in order to evaluate the efficiencies of the models. This started with creating the folds for the dataset based on the speech sentiments to predict. The tree made for the model was then trained to fit the model. Afterwards, the precision, recall, and F1 scores for each class were calculated to assess the false positives and true negatives the model obtained. The results were then pooled into a table for further explanation.

Experimental Results

The results of this experiment provided a success in trying to objectively solve the situation. The algorithms were able to calculate the sentiment score, entropies, and TF_IDF for each tweet, and can be found in raw screenshots displayed in the Appendix.

The support vector machine model proved to show fruitful results. The optimal gamma and cost values are listed below in **Table 1** for the first feature set with entropy and sentiment analysis and in **Table 2** for the TF_IDF section. These tables also include the best performance for both feature sets.

Table 1: Radial Support Vector Machine Set 1 Results

Metric	**Optimal Value**
Cost	10
Gamma	3
Best Performance	0.6649233

Table 2: Radial Support Vector Machine Set 2 Results

Metric	**Optimal Value**
Cost	0.1
Gamma	0.5
Best Performance	0.5835252

The cross-validation errors were also calculated with a tenfold approach. The resulting accuracies for both sets were 69.45% for set one and 57.71% for set two. In addition, the frequency table of predictions were also captured within the program. **Table 3** depicts these values obtained for both feature sets.

Table 3: Frequency Predictions for SVM-radial model, Set 1

Set 1		
Predicted Sentiment	**Truth Sentiment**	**Frequency**
Hate	**Hate**	182
Hate	Offensive	113
Hate	Regular	111
Offensive	Hate	191
Offensive	**Offensive**	98
Offensive	Regular	127
Regular	Hate	192
Regular	Offensive	109
Regular	**Regular**	121
Set 2		
Predicted Sentiment	**Truth Sentiment**	**Frequency**
Hate	**Hate**	183
Hate	Offensive	100
Hate	Regular	117
Offensive	Hate	173
Offensive	**Offensive**	109
Offensive	Regular	112
Regular	Hate	209
Regular	Offensive	111
Regular	**Regular**	130

Based on these frequency tables, a set of recall, precision, and F1-measure scores could be calculated for both feature sets for their SVM modeling approach, displayed below in **Table 4**.

Table 4: Evaluation Metrics for SVM-radial models for both sets

Set 1				
Class	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Hate	51.21%	0.45	0.32	0.37
Offensive	56.59%	0.24	0.32	0.27
Regular	56.67%	0.29	0.34	0.31
Set 2				
	Accuracy	**Precision**	**Recall**	**F1 Score**
Hate	51.85%	0.46	0.32	0.38
Offensive	60.13%	0.28	0.34	0.31
Regular	55.87%	0.29	0.36	0.32

The k-Nearest Neighbors model results for both feature sets are displayed below in **Table 5**. These results include the tested k-values selected with their accuracy values. The frequency tables of their predictions for the most accurate k-values tested are displayed in **Table 6**. The resulting precision, recall, and F1-measure scores were then calculated and listed in **Table 7** for both sets.

Table 5: Accuracy Results for kNN Model Classification, Both Sets

	Accuracy for k = 1	**Accuracy for k = 5**	**Accuracy for k = 20**
Set 1	40%	35%	34%
Set 2	34%	30%	38%

Table 6: Frequency Predictions, Most Accurate kNN models, both sets

Set 1 (k = 1)		
Predicted Sentiment	**Truth Sentiment**	**Frequency**
Hate	**Hate**	14
Hate	Offensive	11
Hate	Regular	11
Offensive	Hate	11
Offensive	**Offensive**	13
Offensive	Regular	8
Regular	Hate	7
Regular	Offensive	12
Regular	**Regular**	13
Set 2 (k = 20)		
Hate	**Hate**	13
Hate	Offensive	10
Hate	Regular	13
Offensive	Hate	14
Offensive	**Offensive**	9
Offensive	Regular	9
Regular	Hate	10
Regular	Offensive	6
Regular	**Regular**	16

Table 7: Evaluation Metrics for Most Accurate kNN Models for Both Sets

Set 1 (k = 1)				
Class	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Hate	60%	0.39	0.44	0.41
Offensive	58%	0.41	0.36	0.48
Regular	62%	0.41	0.41	0.41
Set 2 (k = 20)				
	Accuracy	**Precision**	**Recall**	**F1 Score**
Hate	53%	0.36	0.35	0.36
Offensive	61%	0.28	0.36	0.32
Regular	62%	0.5	0.42	0.46

A neural network method was also applied to make classification models. The resulting prediction frequencies are displayed below in **Table 8**. The evaluation metrics for this method for both sets are captured in **Table 9.**

Table 8: Frequency Predictions, Neural Network Models, both sets

Set 1 (size = 1)		
Predicted Sentiment	**Truth Sentiment**	**Frequency**
Hate	**Hate**	12
Hate	Offensive	134
Hate	Regular	198
Offensive	Hate	15
Offensive	**Offensive**	126
Offensive	Regular	207
Regular	Hate	15
Regular	Offensive	136
Regular	**Regular**	157

Set 2 (size = 2)		
Hate	**Hate**	221
Hate	Offensive	105
Hate	Regular	18
Offensive	Hate	135
Offensive	**Offensive**	196
Offensive	Regular	17
Regular	Hate	149
Regular	Offensive	151
Regular	**Regular**	8

Table 9: Evaluation Metrics for Neural Network Models for Both Sets

Set 1				
Class	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Hate	63.8%	0.035	0.29	0.062
Offensive	50.8%	0.36	0.32	0.34
Regular	44.4%	0.51	0.28	0.36
Set 2				
	Accuracy	**Precision**	**Recall**	**F1 Score**
Hate	59.3%	0.64	0.44	0.52
Offensive	59.2%	0.56	0.43	0.49
Regular	66.5%	0.026	0.19	0.046

Conclusion

There are many conclusions to be made when looking at the results of this experiment between the models used and the feature sets used. To begin, the model with the highest overall accuracy was the neural network model. Between the two sets, it had a total accuracy rate of around 39.5%. It should be noted however that it was more accurate in predicting hate speech based on the second feature set over the first feature set. The k-Nearest Neighbors model had the most consistent accuracy rates based on the two k-values selected, with 40% and 38% for sets one and two respectively. In terms of precision, the neural network also had the overall highest precision values. Once again, it should be noted that the hate class precision value for the first feature set and the regular class precision value for the second feature set were drastically lower compared to all the methods. This could signify that the neural network was not as effective finding hate speech given the first feature set and finding regular speech given the second feature set. In terms of recall, the model that performed the overall best was the k-Nearest Neighbor model. This model's classifying of speech sentiments across both feature sets was the most consistent of all three methods. Lastly, the k-Nearest Neighbor model had the highest F1 scores of all the models used. In all three evaluation metrics, the radial support vector machine provided the lowest amount of correct classifications. As such, this method should be dismissed when classifying hate speech for this experiment. It did have consistent values across all metrics for both feature sets, but it did not outperform the other methods. In addition, this method took the longest to compute, so it is conclusive to say that this method was not efficient for classifying the sentiments of these tweets.

Between the k-Nearest Neighbor and Neural Network classification algorithms, the neutral network algorithm seems to provide the better classification mechanics. While it did not perform well at predicting hate speech based on the first feature set and regular speech based on the second feature set, the fact remains that it provided the best accuracy and precision values compared to the k-Nearest Neighbor method. In addition, the testing and training sets for the k-Nearest Neighbor sets used smaller amounts of data, so it is less reliable when scaled higher. This is because of the specification of k-values as well as how the algorithm cannot accept large amounts of data in RStudio. The most optimal k-value is also more elusive to find than calculating the best neural network size. Overall, the neural network model should be applied in analyzing the sentiment of this set and future sets of tweets based on these feature sets experimented on.

It is very interesting to note how some models were better at predicting certain types of speech over others. For example, the radial support vector machine had the overall highest F1 scores when classifying hate speech in the tweets. On the other hand, the k-Nearest Neighbor model was better at identifying regular speech based on the F1 scores it returned. Lastly, the neural network model was better at recognizing offensive speech in the tweets compared to the other models. It would be thoughtful to research in the future if the models had specific speech sentiment biases when performing their classification algorithms based on the evidence of this experiment.

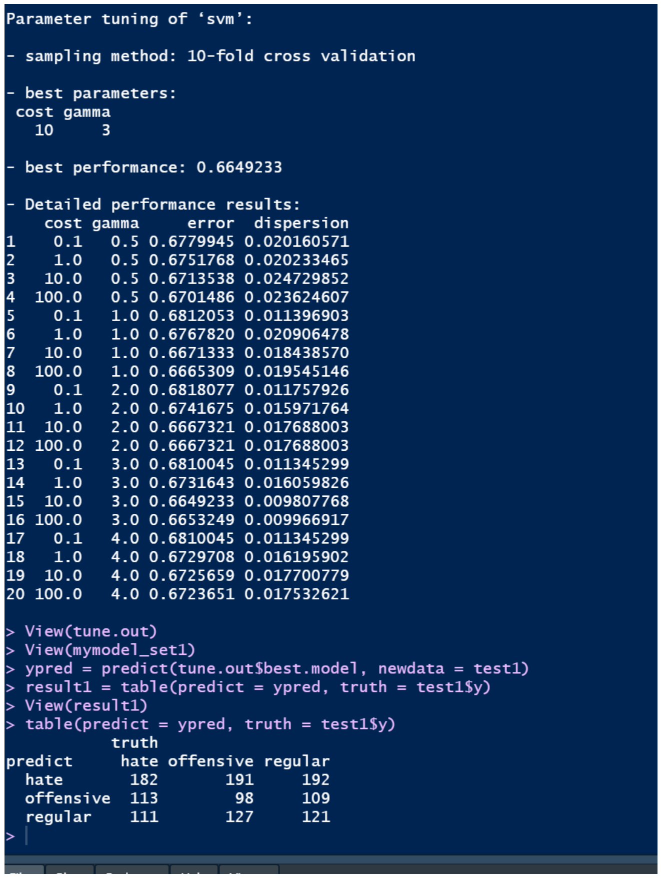
Perhaps one of the most interesting conclusions to be made after the experiment were how well the models were able to predict the speech sentiment based on the feature sets being used. In almost all cases, when the models were tested against the second feature set, the evaluation metrics were higher than those returned from testing of the first feature set. This could be explained by the fact that with fewer variables to model our algorithms on, we are able to make more accurate classifications on our data. However, as future datasets get larger with more variables to analyze, it is more pragmatic to use feature sets that have meaningful and varied feature sets. The first feature set had more variables to model against as well as variables that had more value than the one TF_IDF variable the second feature set had to model against. As such, even though the second feature set returned slightly higher evaluation metrics, it is recommended to use the first feature set to use to train our classification models with. In turn, we will obtain a higher significance of interpretation of the data we classify. In fact, it would be proactive to combine both feature sets and train more classifying algorithms with this larger set of variables to produce even more meaningful results.

References

1. https://stackoverflow.com/questions/17200114/how-to-split-data-into-training-testing-sets-using-sample-function
2. https://rstudio-pubs-static.s3.amazonaws.com/123438_3b9052ed40ec4cd2854b72d1aa154df9.html
3. https://stackoverflow.com/questions/19396947/how-can-i-resolve-the-following-dimension-mismatch-with-rs-k-nearest-neighbors
4. https://www.youtube.com/watch?v=AFg2MvhFeho
5. https://www.heatonresearch.com/2013/06/12/r-classification.html
6. https://www.r-bloggers.com/classification-using-neural-net-in-r/
7. https://rdrr.io/cran/caret/man/recall.html
8. https://stat.ethz.ch/pipermail/r-help/2007-July/137071.html

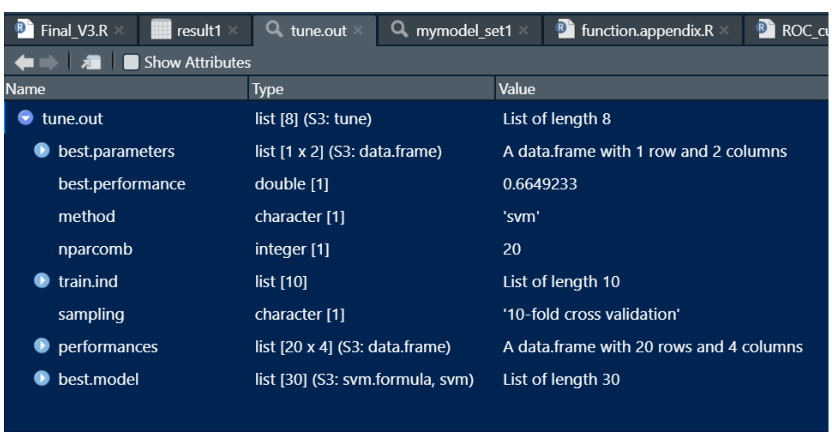
Appendix

Raw Program Screenshots:



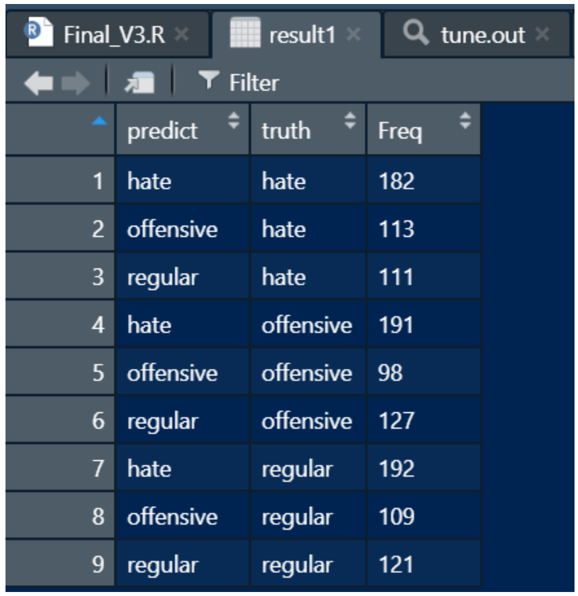
```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10     3

- best performance: 0.6649233

- Detailed performance results:
    cost gamma     error  dispersion
1    0.1   0.5 0.6779945 0.020160571
2    1.0   0.5 0.6751768 0.020233465
3   10.0   0.5 0.6713538 0.024729852
4  100.0   0.5 0.6701486 0.023624607
5    0.1   1.0 0.6812053 0.011396903
6    1.0   1.0 0.6767820 0.020906478
7   10.0   1.0 0.6671333 0.018438570
8  100.0   1.0 0.6665309 0.019545146
9    0.1   2.0 0.6818077 0.011757926
10   1.0   2.0 0.6741675 0.015971764
11  10.0   2.0 0.6667321 0.017688003
12 100.0   2.0 0.6667321 0.017688003
13   0.1   3.0 0.6810045 0.011345299
14   1.0   3.0 0.6731643 0.016059826
15  10.0   3.0 0.6649233 0.009807768
16 100.0   3.0 0.6653249 0.009966917
17   0.1   4.0 0.6810045 0.011345299
18   1.0   4.0 0.6729708 0.016195902
19  10.0   4.0 0.6725659 0.017700779
20 100.0   4.0 0.6723651 0.017532621

> View(tune.out)
> View(mymodel_set1)
> ypred = predict(tune.out$best.model, newdata = test1)
> result1 = table(predict = ypred, truth = test1$y)
> View(result1)
> table(predict = ypred, truth = test1$y)
           truth
predict     hate offensive regular
  hate       182       191     192
  offensive  113        98     109
  regular    111       127     121
>
```

Fig. 1 - Tuning of SVM-Radial Model, Set 1



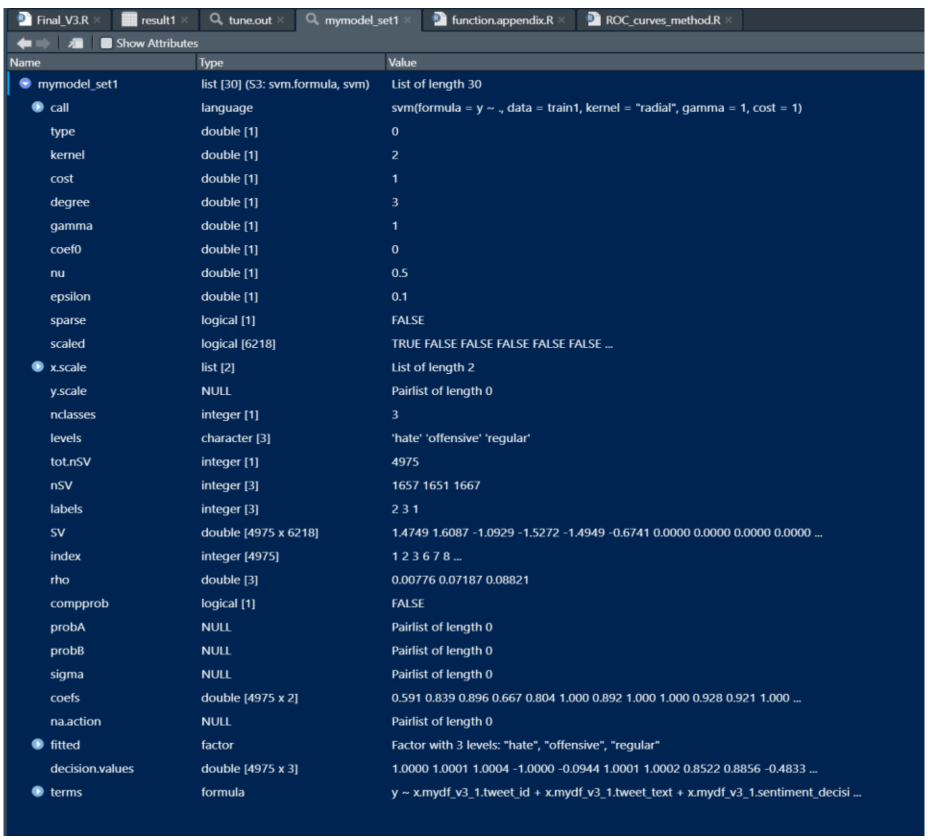
Name	Type	Value
tune.out	list [8] (S3: tune)	List of length 8
best.parameters	list [1 x 2] (S3: data.frame)	A data.frame with 1 row and 2 columns
best.performance	double [1]	0.6649233
method	character [1]	'svm'
nparcomb	integer [1]	20
train.ind	list [10]	List of length 10
sampling	character [1]	'10-fold cross validation'
performances	list [20 x 4] (S3: data.frame)	A data.frame with 20 rows and 4 columns
best.model	list [30] (S3: svm.formula, svm)	List of length 30

Fig. 2 – Tuning Summary of SVM-Radial Model, Set 1



	predict	truth	Freq
1	hate	hate	182
2	offensive	hate	113
3	regular	hate	111
4	hate	offensive	191
5	offensive	offensive	98
6	regular	offensive	127
7	hate	regular	192
8	offensive	regular	109
9	regular	regular	121

Fig. 3 – SVM-Radial Model Prediction Frequencies, Set 1



Name	Type	Value
mymodel_set1	list [30] (S3: svm.formula, svm)	List of length 30
call	language	svm(formula = y ~ ., data = train1, kernel = "radial", gamma = 1, cost = 1)
type	double [1]	0
kernel	double [1]	2
cost	double [1]	1
degree	double [1]	3
gamma	double [1]	1
coef0	double [1]	0
nu	double [1]	0.5
epsilon	double [1]	0.1
sparse	logical [1]	FALSE
scaled	logical [6218]	TRUE FALSE FALSE FALSE FALSE FALSE ...
x.scale	list [2]	List of length 2
y.scale	NULL	Pairlist of length 0
nclasses	integer [1]	3
levels	character [3]	'hate' 'offensive' 'regular'
tot.nSV	integer [1]	4975
nSV	integer [3]	1657 1651 1667
labels	integer [3]	2 3 1
SV	double [4975 x 6218]	1.4749 1.6087 -1.0929 -1.5272 -1.4949 -0.6741 0.0000 0.0000 0.0000 0.0000 ...
index	integer [4975]	1 2 3 6 7 8 ...
rho	double [3]	0.00776 0.07187 0.08821
compprob	logical [1]	FALSE
probA	NULL	Pairlist of length 0
probB	NULL	Pairlist of length 0
sigma	NULL	Pairlist of length 0
coefs	double [4975 x 2]	0.591 0.839 0.896 0.667 0.804 1.000 0.892 1.000 1.000 0.928 0.921 1.000 ...
na.action	NULL	Pairlist of length 0
fitted	factor	Factor with 3 levels: "hate", "offensive", "regular"
decision.values	double [4975 x 3]	1.0000 1.0001 1.0004 -1.0000 -0.0944 1.0001 1.0002 0.8522 0.8856 -0.4833 ...
terms	formula	y ~ x.mydf_v3_1.tweet_id + x.mydf_v3_1.tweet_text + x.mydf_v3_1.sentiment_decisi ...

Fig. 4 - SVM-Radial Model Summary, Set 1

```
> summary(tune.out2)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  0.1   0.5

- best performance: 0.5835252

- Detailed performance results:
    cost gamma     error dispersion
1    0.1   0.5 0.5835252 0.02475225
2    1.0   0.5 0.5845285 0.02774789
3   10.0   0.5 0.5873417 0.03211996
4  100.0   0.5 0.5861357 0.03212660
5    0.1   1.0 0.5883478 0.02426613
6    1.0   1.0 0.5869389 0.02806927
7   10.0   1.0 0.5845289 0.02910744
8  100.0   1.0 0.5845289 0.02910744
9    0.1   2.0 0.6743740 0.01537346
10   1.0   2.0 0.5871417 0.03154170
11  10.0   2.0 0.5845297 0.03238358
12 100.0   2.0 0.5845297 0.03238358
13   0.1   3.0 0.6739715 0.01606393
14   1.0   3.0 0.5998057 0.03072336
15  10.0   3.0 0.5929711 0.02674288
16 100.0   3.0 0.5929711 0.02674288
17   0.1   4.0 0.6731667 0.01764027
18   1.0   4.0 0.6679487 0.02501641
19  10.0   4.0 0.6595081 0.02871736
20 100.0   4.0 0.6595081 0.02871736
```

Fig. 5 - Tuning of SVM-Radial Model, Set 2

Final_V3.R* | tune.out2 | mymodel_set2 | result1 | function.appendix.R | ROC_curves_method.R

Show Attributes

Name	Type	Value
tune.out2	list [8] (S3: tune)	List of length 8
best.parameters	list [1 x 2] (S3: data.frame)	A data.frame with 1 row and 2 columns
best.performance	double [1]	0.5835252
method	character [1]	'svm'
nparcomb	integer [1]	20
train.ind	list [10]	List of length 10
sampling	character [1]	'10-fold cross validation'
performances	list [20 x 4] (S3: data.frame)	A data.frame with 20 rows and 4 columns
best.model	list [30] (S3: svm.formula, svm)	List of length 30

Fig. 6 – Tuning Summary of SVM-Radial Model, Set 2

Final_V3.R* | result2 | tune.

Filter

	predict	truth	Freq
1	hate	hate	183
2	offensive	hate	100
3	regular	hate	117
4	hate	offensive	173
5	offensive	offensive	109
6	regular	offensive	112
7	hate	regular	209
8	offensive	regular	111
9	regular	regular	130

Fig. 7 – SVM-Radial Model Prediction Frequencies, Set 2

Final_V3.R* | mymodel_set2 | result1 | function.appendix.R | ROC_curves_method.R

Show Attributes

Name	Type	Value
mymodel_set2	list [30] (S3: svm.formula, svm)	List of length 30
call	language	svm(formula = y ~ ., data = train2, kernel = "radial", gamma = 1, cost = 1)
type	double [1]	0
kernel	double [1]	2
cost	double [1]	1
degree	double [1]	3
gamma	double [1]	1
coef0	double [1]	0
nu	double [1]	0.5
epsilon	double [1]	0.1
sparse	logical [1]	FALSE
scaled	logical [6216]	TRUE FALSE FALSE FALSE FALSE FALSE ...
x.scale	list [2]	List of length 2
y.scale	NULL	Pairlist of length 0
nclasses	integer [1]	3
levels	character [3]	'hate' 'offensive' 'regular'
tot.nSV	integer [1]	4975
nSV	integer [3]	1673 1679 1623
labels	integer [3]	1 2 3
SV	double [4975 x 6216]	-1.3498 -0.5699 0.5646 -0.8070 1.6460 -1.0356 0.0000 0.0000 0.0000 0.0000 ...
index	integer [4975]	1 8 9 11 15 17 ...
rho	double [3]	-0.0379 0.0514 0.0459
compprob	logical [1]	FALSE
probA	NULL	Pairlist of length 0
probB	NULL	Pairlist of length 0
sigma	NULL	Pairlist of length 0
coefs	double [4975 x 2]	1.000 1.000 1.000 1.000 1.000 1.000 0.538 0.467 0.323 1.000 1.000 0.403 ...
na.action	NULL	Pairlist of length 0
fitted	factor	Factor with 3 levels: "hate", "offensive", "regular"
decision.values	double [4975 x 3]	0.5385 -0.9999 0.0836 -1.0000 0.3535 -0.9998 0.9999 0.6690 -0.2259 0.2048 ...
terms	formula	y ~ x.mydf_v3_2.tweet_id + x.mydf_v3_2.tweet_text + x.tfidf_words.total_tf_idf

Fig. 8 - SVM-Radial Model Summary, Set 2

	Error 10CV Set1	Error 10CV Set2
1	0.6943225	0.5770995

Fig. 9 - SVM-Radial Model 10CV Error Calculations

	knn.1set1	knn_sentiment_test1	Freq
1	hate	hate	14
2	offensive	hate	11
3	regular	hate	11
4	hate	offensive	11
5	offensive	offensive	13
6	regular	offensive	8
7	hate	regular	7
8	offensive	regular	12
9	regular	regular	13

Fig. 10 – kNN-1 Model Prediction Frequencies Summary, Set 1

	knn.5set1	knn_sentiment_test1	Freq
1	hate	hate	12
2	offensive	hate	14
3	regular	hate	10
4	hate	offensive	10
5	offensive	offensive	13
6	regular	offensive	9
7	hate	regular	7
8	offensive	regular	15
9	regular	regular	10

Fig. 11 – kNN-5 Model Prediction Frequencies Summary, Set 1

	knn.20set1	knn_sentiment_test1	Freq
1	hate	hate	10
2	offensive	hate	12
3	regular	hate	14
4	hate	offensive	11
5	offensive	offensive	12
6	regular	offensive	9
7	hate	regular	11
8	offensive	regular	9
9	regular	regular	12

Fig. 12 – kNN-20 Model Prediction Frequencies Summary, Set 1

	knn.1set2	knn_sentiment_test2	Freq
1	hate	hate	9
2	offensive	hate	13
3	regular	hate	14
4	hate	offensive	7
5	offensive	offensive	13
6	regular	offensive	12
7	hate	regular	11
8	offensive	regular	9
9	regular	regular	12

Fig. 13 – kNN-1 Model Prediction Frequencies Summary, Set 2

	knn.5set2	knn_sentiment_test2	Freq
1	hate	hate	7
2	offensive	hate	15
3	regular	hate	14
4	hate	offensive	10
5	offensive	offensive	12
6	regular	offensive	10
7	hate	regular	10
8	offensive	regular	11
9	regular	regular	11

Fig. 14 – kNN-5 Model Prediction Frequencies Summary, Set 2

	knn.20set2	knn_sentiment_test2	Freq
1	hate	hate	13
2	offensive	hate	10
3	regular	hate	13
4	hate	offensive	14
5	offensive	offensive	9
6	regular	offensive	9
7	hate	regular	10
8	offensive	regular	6
9	regular	regular	16

Fig. 15 – kNN-20 Model Prediction Frequencies Summary, Set 2

	Accuracy for k=1	Accuracy for k=5	Accuracy for k=20
1	40	35	34

Fig. 16 – kNN Model Accuracy Summary, Set 1

	Accuracy for k=1	Accuracy for k=5	Accuracy for k=20
1	34	30	38

Fig. 17 – kNN Model Accuracy Summary, Set 2

	Var1	Var2	Freq
1	0	0	12
2	0.5	0	134
3	1	0	198
4	0	0.5	15
5	0.5	0.5	126
6	1	0.5	207
7	0	1	15
8	0.5	1	136
9	1	1	157

Fig. 18 – Neural Network Model Prediction Frequencies Summary, Set 1

	Var1	Var2	Freq
1	0	0	221
2	0.5	0	105
3	1	0	18
4	0	0.5	135
5	0.5	0.5	196
6	1	0.5	17
7	0	1	149
8	0.5	1	151
9	1	1	8

Fig. 19 – Neural Network Model Prediction Frequencies Summary, Set 2

```
> tmodel_set1 = tune.nnet(y_new~senty+score, data = nnet_ds1, size = 1:10)
> summary(tmodel_set1)

Parameter tuning of 'nnet':

- sampling method: 10-fold cross validation

- best parameters:
 size
    1

- best performance: 0.1669627

- Detailed performance results:
   size     error  dispersion
1     1 0.1669627 0.003760382
2     2 0.1670558 0.003926611
3     3 0.1672249 0.003895099
4     4 0.1672936 0.004002691
5     5 0.1674042 0.003981957
6     6 0.1675531 0.003971194
7     7 0.1674954 0.003849873
8     8 0.1675934 0.003889413
9     9 0.1675925 0.004006882
10   10 0.1675244 0.003954181
```

Fig. 20 – Tuning Neural Network Model, Set 1

Final_V3.R × | tmodel_set1 ×

Show Attributes

Name	Type	Value
tmodel_set1	list [8] (S3: tune)	List of length 8
best.parameters	list [1 x 1] (S3: data.frame)	A data.frame with 1 row and 1 column
best.performance	double [1]	0.1669627
method	character [1]	'nnet'
nparcomb	integer [1]	10
train.ind	list [10]	List of length 10
sampling	character [1]	'10-fold cross validation'
performances	list [10 x 3] (S3: data.frame)	A data.frame with 10 rows and 3 columns
best.model	list [18] (S3: nnet.formula, nnet)	List of length 18

Fig. 21 – Tuning Neural Network Model Summary, Set 1

```
> tmodel_set2 = tune.nnet(y_new~tfidf_scaled, data = nnet_ds2, size = 1:10)
> summary(tmodel_set2)

Parameter tuning of 'nnet':

- sampling method: 10-fold cross validation

- best parameters:
 size
    3

- best performance: 0.1600627

- Detailed performance results:
   size     error  dispersion
1     1 0.1600823 0.004895097
2     2 0.1600804 0.005272889
3     3 0.1600627 0.005390954
4     4 0.1601204 0.005431733
5     5 0.1602622 0.005332256
6     6 0.1601772 0.005412867
7     7 0.1602181 0.005405760
8     8 0.1602787 0.005380644
9     9 0.1602965 0.005367814
10   10 0.1602400 0.005407138
```

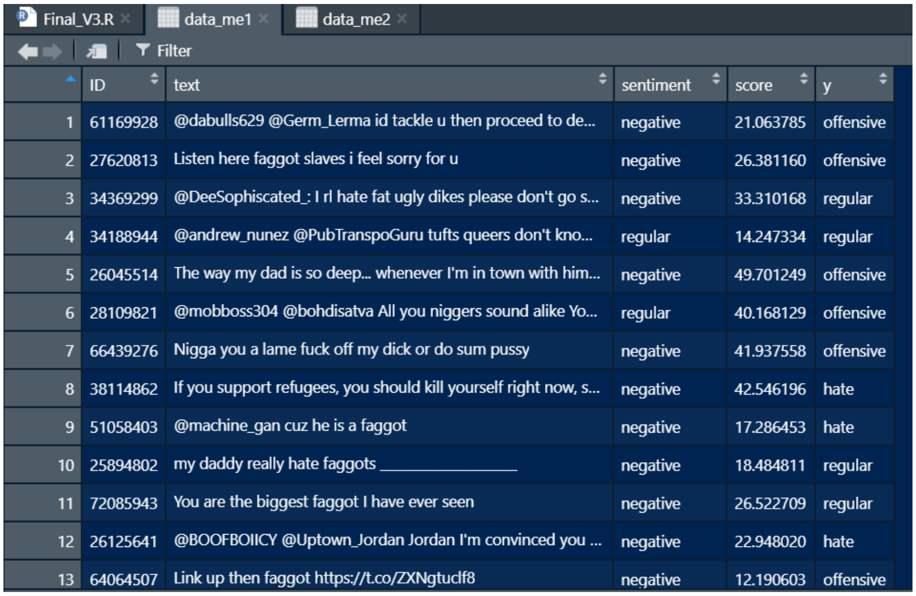
Fig. 22 – Tuning Neural Network Model, Set 2

Final_V3.R | tmodel_set1 | tmodel_set2

Show Attributes

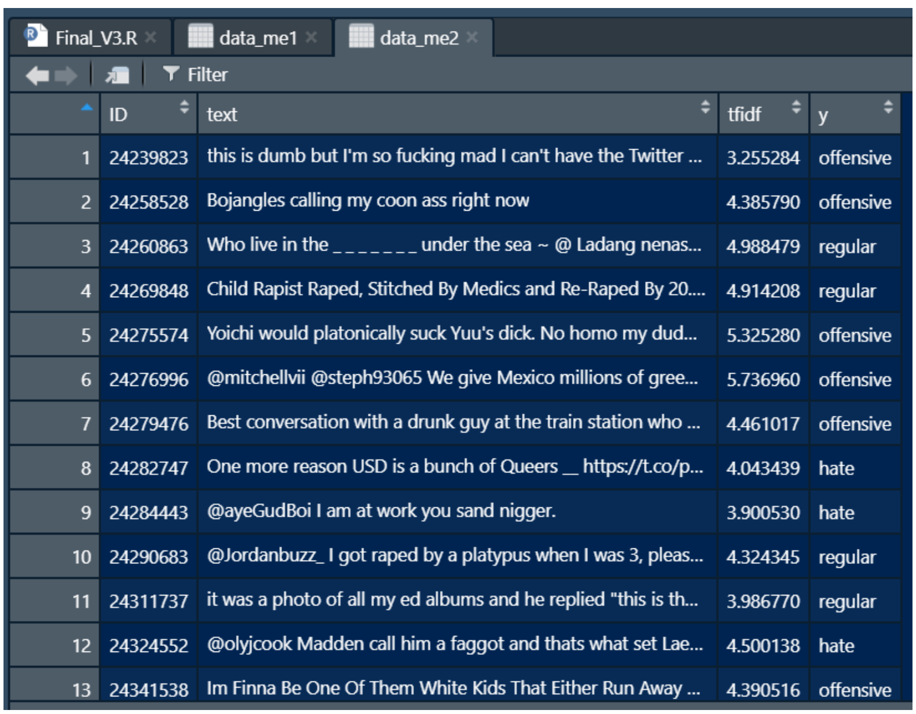
Name	Type	Value
tmodel_set2	list [8] (S3: tune)	List of length 8
best.parameters	list [1 x 1] (S3: data.frame)	A data.frame with 1 row and 1 column
best.performance	double [1]	0.1600627
method	character [1]	'nnet'
nparcomb	integer [1]	10
train.ind	list [10]	List of length 10
sampling	character [1]	'10-fold cross validation'
performances	list [10 x 3] (S3: data.frame)	A data.frame with 10 rows and 3 columns
best.model	list [18] (S3: nnet.formula, nnet)	List of length 18

Fig. 23 – Tuning Neural Network Model Summary, Set 2



	ID	text	sentiment	score	y
1	61169928	@dabulls629 @Germ_Lerma id tackle u then proceed to de...	negative	21.063785	offensive
2	27620813	Listen here faggot slaves i feel sorry for u	negative	26.381160	offensive
3	34369299	@DeeSophiscated_: I rl hate fat ugly dikes please don't go s...	negative	33.310168	regular
4	34188944	@andrew_nunez @PubTranspoGuru tufts queers don't kno...	regular	14.247334	regular
5	26045514	The way my dad is so deep... whenever I'm in town with him...	negative	49.701249	offensive
6	28109821	@mobboss304 @bohdisatva All you niggers sound alike Yo...	regular	40.168129	offensive
7	66439276	Nigga you a lame fuck off my dick or do sum pussy	negative	41.937558	offensive
8	38114862	If you support refugees, you should kill yourself right now, s...	negative	42.546196	hate
9	51058403	@machine_gan cuz he is a faggot	negative	17.286453	hate
10	25894802	my daddy really hate faggots _______________	negative	18.484811	regular
11	72085943	You are the biggest faggot I have ever seen	negative	26.522709	regular
12	26125641	@BOOFBOIICY @Uptown_Jordan Jordan I'm convinced you ...	negative	22.948020	hate
13	64064507	Link up then faggot https://t.co/ZXNgtucIf8	negative	12.190603	offensive

Fig. 24 – Snapshot of Set 1 Data Frame



	ID	text	tfidf	y
1	24239823	this is dumb but I'm so fucking mad I can't have the Twitter ...	3.255284	offensive
2	24258528	Bojangles calling my coon ass right now	4.385790	offensive
3	24260863	Who live in the _ _ _ _ _ _ under the sea ~ @ Ladang nenas...	4.988479	regular
4	24269848	Child Rapist Raped, Stitched By Medics and Re-Raped By 20....	4.914208	regular
5	24275574	Yoichi would platonically suck Yuu's dick. No homo my dud...	5.325280	offensive
6	24276996	@mitchellvii @steph93065 We give Mexico millions of gree...	5.736960	offensive
7	24279476	Best conversation with a drunk guy at the train station who ...	4.461017	offensive
8	24282747	One more reason USD is a bunch of Queers __ https://t.co/p...	4.043439	hate
9	24284443	@ayeGudBoi I am at work you sand nigger.	3.900530	hate
10	24290683	@Jordanbuzz_ I got raped by a platypus when I was 3, pleas...	4.324345	regular
11	24311737	it was a photo of all my ed albums and he replied "this is th...	3.986770	regular
12	24324552	@olyjcook Madden call him a faggot and thats what set Lae...	4.500138	hate
13	24341538	Im Finna Be One Of Them White Kids That Either Run Away ...	4.390516	offensive

Fig. 25 – Snapshot of Set 2 Data Frame

Equations Used:

Eq. 1 $Recall = \frac{TP}{TP + FN}$

Eq. 2 Precision $= \frac{TP}{TP + FP}$

Eq. 3 F1 Score $= \frac{2 * Precision * Recall}{Precision + Recall}$

Or

$$\frac{(1 + \beta^2) * Precision * Recall}{\beta^2 * Precision + Recall}$$

RStudio Packages/Libraries Used:

- library(tidytext)
- library(tidyr)
- library(dplyr)
- library(stringr)
- library(caret)
- library(ggplot2)
- library(sentimentr)
- library(tm)
- library(gmodels)
- library(caTools)
- library(e1071)
- library(ggplot2)
- library(caret)
- library(tree)
- library(party)
- library(class)
- library(MASS)
- library(nnet)

Calculations of Evaluation Metrics (from https://confusionmatrixonline.com/)

SVM – Set 1

	Class 1	Class 2	Class 3
Class 1	182	113	111
Class 2	191	98	127
Class 3	192	109	121
Total for Class	565	320	359

No cell selected

Results

TP: 401
Overall Accuracy: 32.23%

Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	565	406	51.21%	0.45	0.32	0.37
2	320	416	56.59%	0.24	0.31	0.27
3	359	422	56.67%	0.29	0.34	0.31

SVM – Set 2

	Class 1	Class 2	Class 3
Class 1	183	100	117
Class 2	173	109	112
Class 3	209	111	130
Total for Class	565	320	359

No cell selected

Results

TP: 422
Overall Accuracy: 33.92%

Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	565	400	51.85%	0.46	0.32	0.38
2	320	394	60.13%	0.28	0.34	0.31
3	359	450	55.87%	0.29	0.36	0.32

kNN-1 – Set 1

	Class 1	Class 2	Class 3
Class 1	14	11	11
Class 2	11	13	8
Class 3	7	12	13
Total for Class	32	36	32

No cell selected

Results

TP: 40
Overall Accuracy: 40%

Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	32	36	60%	0.39	0.44	0.41
2	36	32	58%	0.41	0.36	0.38
3	32	32	62%	0.41	0.41	0.41

kNN-20 – Set 2

	Class 1	Class 2	Class 3
Class 1	13	10	13
Class 2	14	9	9
Class 3	10	6	16
Total for Class	37	25	38

No cell selected

Results

TP: 38
Overall Accuracy: 38%

Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	37	36	53%	0.36	0.35	0.36
2	25	32	61%	0.28	0.36	0.32
3	38	32	62%	0.50	0.42	0.46

NNET – Set 1

	Class 1	Class 2	Class 3
Class 1	12	134	198
Class 2	15	126	207
Class 3	15	136	157
Total for Class	42	396	562

No cell selected

Results

TP: 295
Overall Accuracy: 29.5%

Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	42	344	63.8%	0.035	0.29	0.062
2	396	348	50.8%	0.36	0.32	0.34
3	562	308	44.4%	0.51	0.28	0.36

NNET – Set 2

	Class 1	Class 2	Class 3
Class 1	221	105	18
Class 2	135	196	17
Class 3	149	151	8
Total for Class	505	452	43

No cell selected

Results

TP: 425
Overall Accuracy: 42.5%

Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	505	344	59.3%	0.64	0.44	0.52
2	452	348	59.2%	0.56	0.43	0.49
3	43	308	66.5%	0.026	0.19	0.046