HATE SPEECH DETECTION IN TWEETS USING MACHINE LEARNING MODELS

Pardeep Kumar

Mentor: Dipanjan Sarkar

Springboard Career Science Data Track

MILESTONE REPORT 1

CAPSTONE PROJECT 2

TABLE OF CONTENTS

1 Problem Statement 3
2 Data Mining 4
2.1 Data Mining 4
2.2 Data Preprocessing 6
3 Word Vectors 7
3.1 Bag-of-Words 7
4 Baseline models 9
4.1 Majority Class Baseline Model 9
4.1.1 Performance Measures 10
4.1.2 Accuracy 11
4.1.3 Precision 11
4.1.4 Recall 11
4.1.5 F1-Score 12
4.1.6 AUC 12
4.2 Random Forest Classifiers 13
4.2.1 TF-IDF Vector 13
4.2.2 Random Forests 14

1 PROBLEM STATEMENT

The main objective of this report was to evaluate the performance of the neural networks in detecting weather a tweet is hateful, offensive, or neither, against decision tree models. Text classification is subjective, i.e., one user can find the comment offensive while it might be neutral for another individual. Due to such issues, online communication platforms such as Facebook, Twitter, often have trouble on which side to support where free speech is protected.

One obvious solution could be to mark all the tweets, comments, reviews, as hateful/offensive with a high frequency of abusive language. However, detecting profanity in the text can be challenging as the users often obfuscate the correct spellings such as a$$hole and b!! tch. A better approach would be to train a machine learning model to recognize structural patterns exhibited by the text documents belonging to a particular class.

Despite the fame and popularity, the neural networks have struggled in performance when compared with statistical models such as decisions trees. A recent empirical study [1], showed that in higher-dimensional problems, the random forest classifiers consistently outperformed the neural networks. Decision tree models require heavy preprocessing of the text data before training the classifiers. In comparison, neural networks can perform well on features with an intricate structure. Having a classifier that can perform well without too much computational cost can help curb the offensive/hateful language to appear on online platforms.

This report details the data mining, preprocessing, and baseline model selection for the project.

2 Data Mining

A text data set is a collection of data objects, which in turn are defined by their attributes that reflect the essential characteristics of objects. The Twitter dataset had two types of attributes:

1. Categorical or qualitative attributes such as the class of each tweet (hate, offensive or neutral), and
2. Numeric or quantitative characteristics such as the length of each tweet.

Most machine learning tasks start with:

1. Collection of the raw data, and
2. Preprocessing the raw data to make for analysis.

2.1 Data Mining

The dataset was a collection of 24,784 manually labeled tweets by CrowdFlower [2] users as *hate_speech*, *offensive_langauge*, or *neither* [3]. Table 2.1shows the first two rows of raw data. Out of the six columns in the table, four fields were dropped, keeping only columns labeled *class* and *tweet*.

The class field contained the majority votes a tweet received from the CrowdFlower users out of 0 – hate speech, 1 – offensive speech, and 2 – neutral (neither). As shown in Figure 2.1, the tweet count per class was highly imbalanced with 6%, 77%, and 17% tweets in the hate, offensive, and neutral class, respectively.

Model accuracy score is the standard performance measures in statistics for comparing two classifiers. In the multiclass classification with imbalanced class distribution, relying solely on the accuracy score could lead to an inferior model selection. For classifications tasks, a confusion matrix compares the count of actual labels and predicted labels. Figure 2.2 shows a confusion matrix for the binary text classification problem (offensive or not offensive), and Table 2.2 lists the four standard performance measures.

Note that depending on the task, the invariance of the performance measure may be beneficial or detrimental. A performance measure is invariant if changes in the confusion matrix have no impact on its value. There are two categories of performance measures in case of the multiclass classification problems:

1. *micro-averaging* which sums the count of confusion matrix components to obtain the cumulative values, and
2. *macro-averaging* which computes a simple average over all classes.

A micro-averaged metric is the measure of effectiveness on broad categories, whereas macro-averaged values give a sense of efficacy on individual levels.

Table 2.1 Raw dataset for tweet classification

	count	**Hate speech**	**Offensive language**	**neither**	**class**	**tweet**
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. &amp; as a...
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!

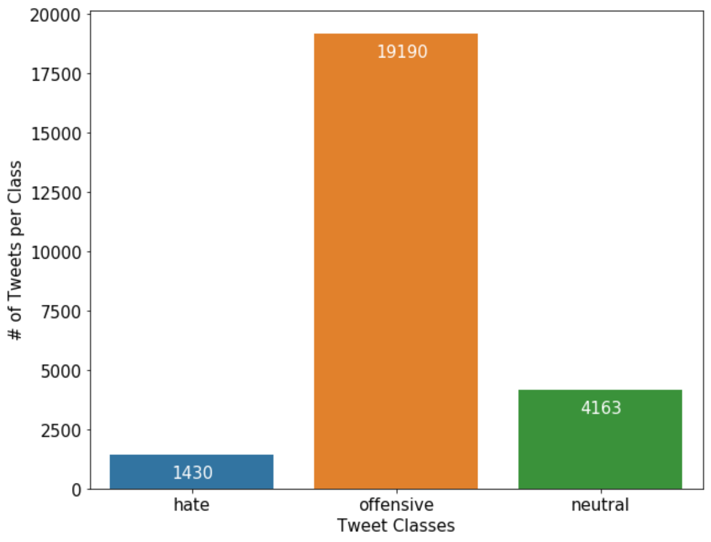
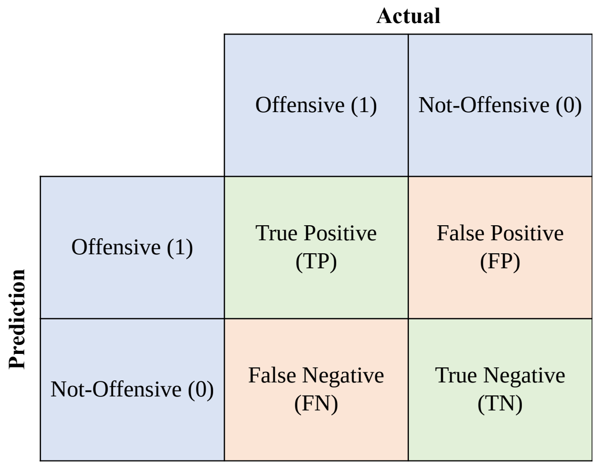


Figure 2.1. Tweet count per class



		Actual	
		Offensive (1)	Not-Offensive (0)
Prediction	Offensive (1)	True Positive (TP)	False Positive (FP)
	Not-Offensive (0)	False Negative (FN)	True Negative (TN)

Figure 2.2 Confusion matrix for binary classification

Table 2.2 Performance measures for binary classification

Accuracy	$\frac{(TP+TN)}{(TP+FP+FN+TN)}$	Accuracy score measures the overall effectiveness of a classifier and is most effective when the class size is relatively balanced.
Precision	$\frac{TP}{(TP+FP)}$	The precision score measures the exactness of the classifier. Precision is high if the number of true positives is high, and false positives are low.
Recall (Sensitivity)	$\frac{TP}{(TP+FN)}$	Recall measures the completeness or sensitivity of the classifier. High recall means that the false-negative count is low and true positives are high.
F1-score	$\frac{2TP}{(2TP+FP+FN)}$	Recall and precision are often at odds. F1-score combines the two by taking their weighted harmonic mean.

2.2 DATA PREPROCESSING

The raw tweets were messy, especially those that belonged to the hate and offensive categories, with poor spelling, grammar, and sentence structure. Often extensive data cleaning is necessary before developing any text classifier. The raw data file was uploaded in the UTF-8 Unicode (byte per character) format preserving the unknown words that are often modified if loaded using ASCII format. The text normalization methods such as stemming or lemmatization were not part of preprocessing. It was done to study how well different classifiers can perform on the moderately processed text.

Following features were removed from the tweet during preprocessing:

1. Username: Usernames were limited to a maximum length of 15 characters after '@' symbol.
2. Web address: Web addresses started either with "*https://*" or directly as "*www*...".
3. Retweet handle: Keyword 'RT' short for re-tweet.
4. Punctuations: Punctuations are the most common feature in the online texts and do not convey any meaning to the machine learning classifiers.
5. Emoticons: Emoji could be useful in the right context but, dropped for this study.
6. Numbers: Numeric data did not add to any text classification relevance.
7. Stop words: Stop words are common words with high occurrence but carry little information for the text classification tasks, e.g., words like *a, an and the*.

All uppercase characters were converted to the lowercases. Figure 2.3 shows the character length distribution of the *raw* and *cleaned* tweets. Maximum character length in the unprocessed dataset was 754 characters exceeding the twitter limit of 140 characters (now increased to 250) because of the twitter emojis in the text. Emoticons are extremely popular in online texts, and their UTF-codes expand to long character strings in datafiles. However, for the current study, they were not included in the clean tweet texts.

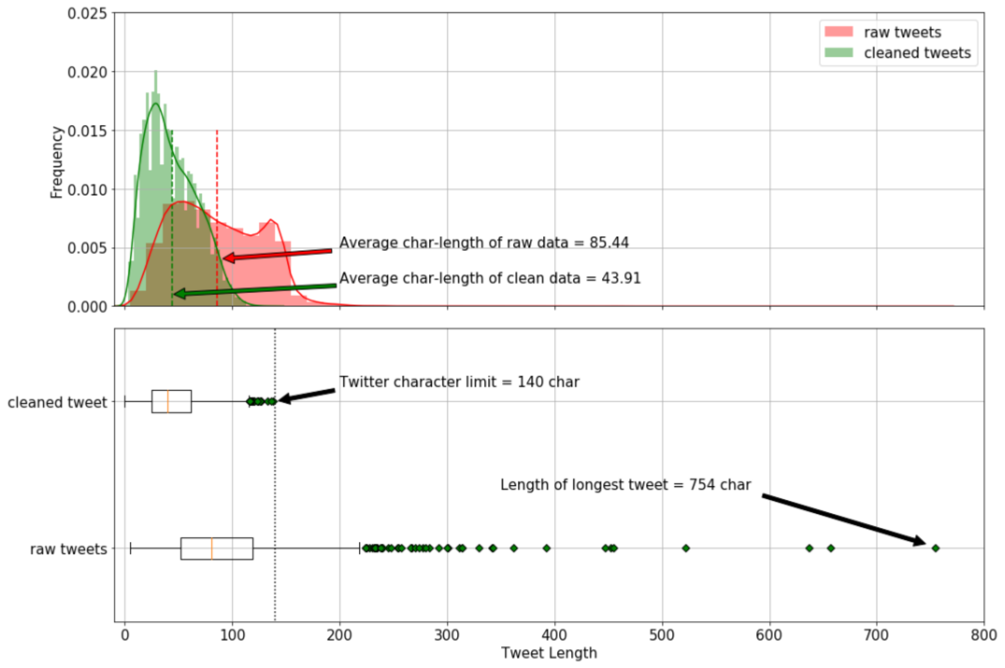


Figure 2.3 Character length comparison of raw- and clean-tweets.

3 WORD VECTORS

A text document must be first converted into a numeric format, as machine learning algorithms cannot process the text format (strings). Tokenization is the process of splitting the text documents into smaller units called tokens, which can then be assigned specific numeric vectors. A token can be an individual character, a word, or an N-gram. N-grams are groups of overlapping consecutive characters or words. The most common form of tokens in natural language processing are the words. The machine learning models are trained to map words to vectors such that vectors of words with similar meaning, are closer. Cosine of angle between two vectors is a good measure to test word similarity. For any two random vectors, say $x_i, x_j \in \mathcal{R}^n$ the cosine similarity is computed as the dot product of unit vectors of x_i and x_j,

$$Similarity_{cosine} = \cos\theta = \frac{x_i}{\|x_i\|_2} \cdot \frac{x_j}{\|x_j\|_2}$$

If the words are the points in the vector space, words with similar meaning are "closer," and the magnitude of the cosine similarity measure is closer to unity. On the other hands, for the two uncorrelated (out-of-context) words, the cosine similarity measure would be small or closer to zero.

3.1 BAG-OF-WORDS

Bag-of-word models hypothesize that relevance of a document to a specific query depends on the word frequency in the document [4]. Bag-of-words model *represent*s a word (token) as a one-hot encoded vector. For example, consider a simple expression *"My dog is always excited and joyful*." Figure 3.1 shows the one-hot encoded representation of the words. Each word in Figure 3.1 is a 7-dimensional unit vector.

$$\begin{array}{ccccccc}
\text{My} & \text{dog} & \text{is} & \text{always} & \text{excited} & \text{and} & \text{joyful} \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

Figure 3.1. One-hot encoding

As the size of vocabulary increases, so does the length of each one-hot vector. Managing large scale sparse vectors (matrices) can be computationally expensive and often impossible. Also, the bag-of-word vectorization technique is not order-preserving, and the meaningful structure of the sentence is lost when transformed into one-hot vectors. In Figure 3.1, the words '*excited*' and '*joyful*' are similar in context but have zero cosine similarity, which would affect the ability of an algorithm to learn the correct document context.

Figure 3.1 is a simple example where each word occurs only once in the text (document). Large documents comprise of hundreds of words repeated multiple time. For example, the word occurs in the context of almost every noun and has high word-frequency. One approach is to replace the indicator (0,1) for word occurrence with the (0, word count). Another approach is to create a co-occurrence matrix. Co-occurrence matrix counts the number of times (frequency) a word appears in a document with other words in the corpus. For larger document space, term-document matrix, where each entry is the count of the target word in different documents, is more informative than the (word-word) co-occurrence matrix.

Stop words (e.g., *a the, of*) occur more frequently in the documents and cause skewness in the co-occurrence matrix without adding any relevant information. On the one hand, the count of more frequently co-occurring words is crucial, and on the other hand, words that are more frequent lack of any useful information. **TF-IDF** algorithm can balance the two paradox to obtain an updated new-matrix. Each entry in the updated matrix is the product of the term-frequency and the inverse document frequency,

$$C_{t,d}^{updated} = tf_{t,d} \times idf_{t,d}$$

Term frequency is obtained by logarithmic transformation (base 10) of the entries of the co-occurrence matrix of a given document ($C_{t,d}$), such that words that occur more frequently receive a higher penalty.

$$tf_{t,d} = \begin{cases} 1 + \log_{10} C_{t,d}\,, if\ C_{t,d} > 0 \\ 0 \qquad\qquad\qquad\quad otherwise \end{cases}$$

The inverse document frequency (idf) assign relative importance to more discriminative words, words that might appear more often in one document but sparsely in the rest of the text-corpus. A large document set is also log-transformed to bound the computed values,

$$idf_{t,d} = \log_{10}\left(\frac{size\ of\ text\ corpus}{number\ of\ documents\ the\ term\ t\ occurs}\right)$$

TF-IDF vector model can capture similarities between words as they focus heavily on target-context word occurrence. However, the issue of large vector (matrix) size remains a challenge as online text data usually has a large vocabulary. The idea of TF-IDF can be applied to characters or word N-grams.

Figure 3.2 shows the distribution of word-counts per tweet using strip and violin plots. The maximum length of tweets in the training set was 26 words (far right green zone), and the median and average word count was 5 and 7, respectively. Note that the plot is based on the training set vocabulary.

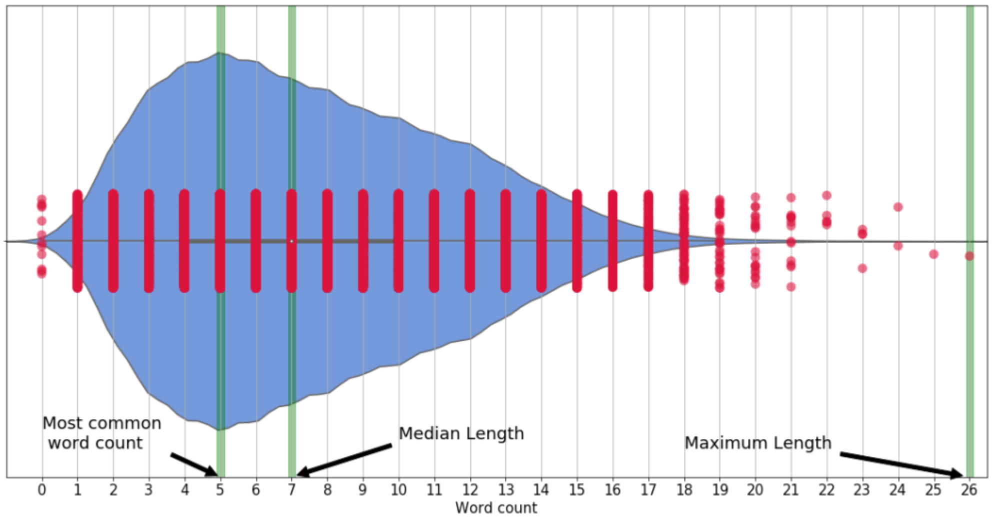


Figure 3.2. Tweet length (word count)

4 BASELINE MODELS

A famous aphorism in statistical learning is "all models are wrong, but some are useful'. Thus, a model can only be evaluated relative terms to another due to lack of an absolute measure in statistics. A baseline model serves as a benchmark against which the performance of other classifiers can be compared. Characteristics such as simple execution and interpretation usually decide a baseline model.

In the text classification problems, the dataset consists of the documents (di) from some document space ($\mathcal{D}$), labeled as one of the classes from a fixed class set, $\mathcal{C} = \{c_1, c_2, \dots c_N\}$. In the supervised learning, the task of a classifier (Ψ) is to find a classification function (ψ) that can accurately map each document to the corresponding class-label, $\psi : \mathcal{D} \rightarrow \mathcal{C}$. This report only presents one-on-one mapping, also called *multiclass classification* problem, where are given text input belongs to only one particular class and all classes are independent of each other. Another classification problem called *multilabel text classification* is one-to-many where a text document can have multiple labels. The goal of machine learning algorithms is to train a classifier on a training dataset and evaluate its performance on the test dataset (data set that the classifier has never *seen* before).

4.1 MAJORITY CLASS BASELINE MODEL

In an imbalanced class dataset, the Majority Class Baseline (MCB) model, identifies the dominant class is the training data set and assigns that class as the prediction to every input in the test set. MCB models are simple, fast to execute and are used in practice when state-of-the-art models are unavailable (e.g., naïve prediction models in time series analysis).

Figure 4.1 shows the histogram with individual category counts in the training and test data set. The category class *hate, offensive and neutral* were assigned class labels of *0, 1 and 2*, respectively. The relative distribution (class ratios) of tweets in the training and test datasets was almost the same. Models were trained and tested for their performance on 20,000 and 4,783 tweets, respectively.

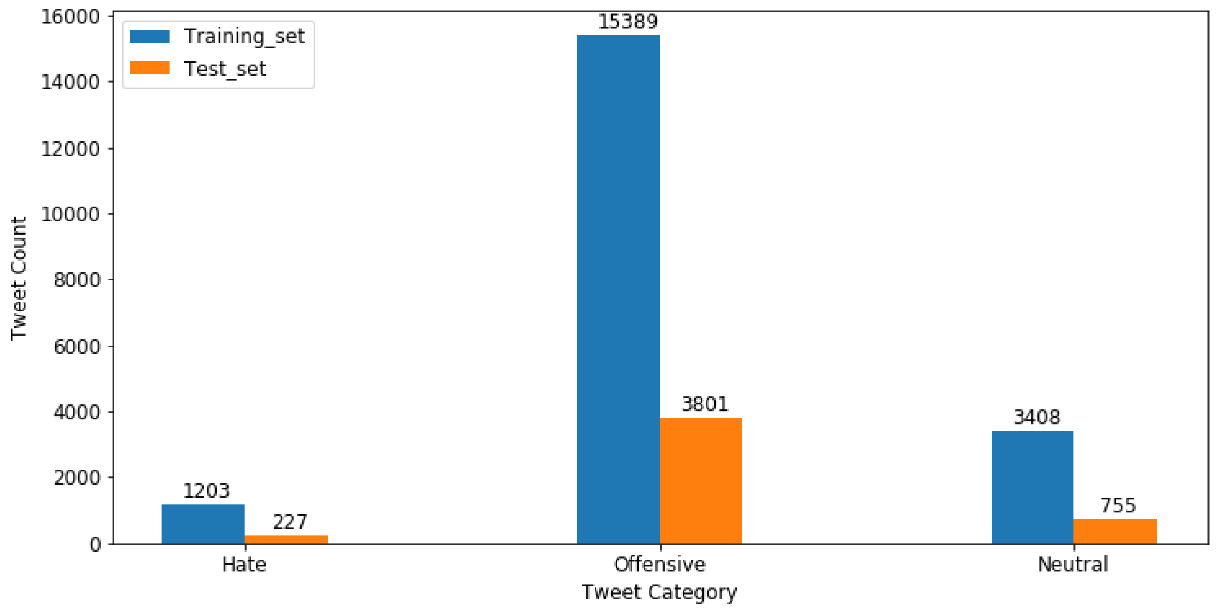


Figure 4.1 Tweet count (per class) in training and test set

4.1.1 Performance Measures

MCB model provided a baseline worst-case scenario where the predictions were guessed based on labels rather than the features of the text. The offensive class was dominant in the training set and was selected as the model prediction for any input.

Table 4.1 and Figure 4.2 shows the multiclass confusion matrix computed for the test dataset. A multiclass confusion matrix has a total C sub-matrices, where C is the total number of classes. The Twitter data set had three submatrices, one for each of the hate, offensive, and neutral class. The submatrices in Figure 4.2 were computed by reducing the multiclass classification problem to a simple binary classification, where tweet either belongs to a particular class or not. For example, the first submatrix was for the binary classification of tweets either in hate class or not-hate (either offensive or neutral). MCB model labeled all tweets as offensive (not hate category); thus the model correctly identifies true negative, i.e., the 4,556 not-hate tweets (3801 in offensive and 755 in neutral class) were correctly labeled. Since none of the tweets were labeled hate, the number of true positives and false positives were zero. Total of 227 tweets was falsely labeled as offensive (or negative in binary terms) with hate class as their correct label.

Table 4.1 Per-class correct and incorrect classification counts

	Hate	**Offensive**	**Neutral**
True Negative	4556	0	4028
False Negative	0	982	0
False Positive	227	0	755
True Positive	0	3801	0

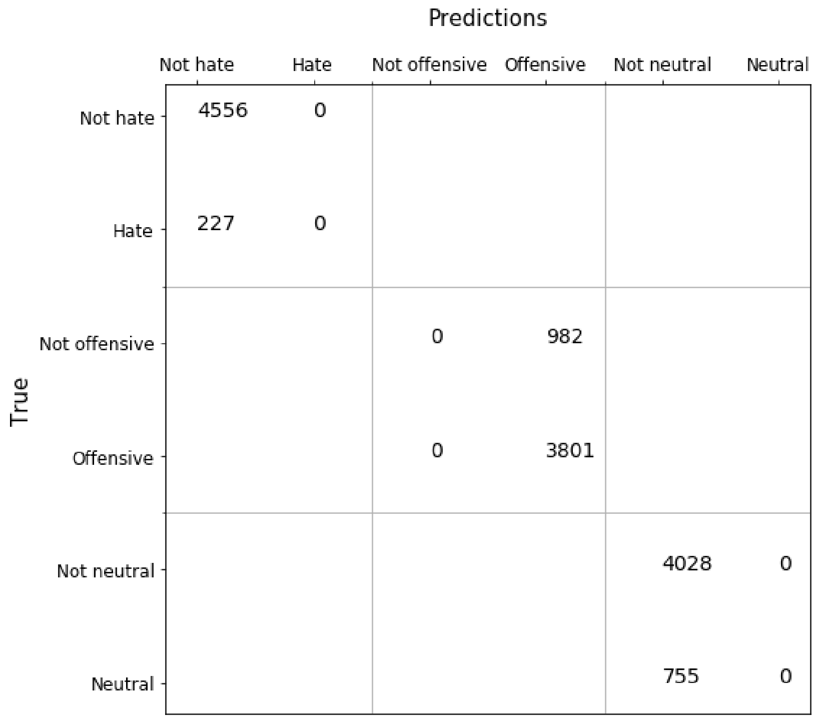


Figure 4.2 Multilabel confusion matrix for test data

4.1.2 Accuracy

The average accuracy represented the effectiveness of each classifier average over the total number of classes. For the balanced dataset, accuracy represents the overall effectiveness of the classifier. The average accuracy of test predictions was computed as below:

$$Accuracy_{average}^{test} = \frac{1}{3}\sum_{class=0}^{2}\left(\frac{TP_{class}+TN_{class}}{TP_{class}+TN_{class}+FP_{class}+FN_{class}}\right)$$

$$Accuracy_{average}^{test} = \frac{1}{3}\left(\frac{0+4556}{0+4556+0+227}+\frac{3801+0}{3801+0+982+0}+\frac{0+4028}{0+4028+0+755}\right)$$

$$Accuracy_{average}^{test} = \frac{1}{3}(0.95+0.79+0.84) = 0.86$$

Thus, the MCB model was 86% accurate rather high number. Thus. The model accuracy was not a reliable performance measure in the case of imbalanced classes (section 2.1).

4.1.3 Precision

Precision represents the agreement between the classifier and the dataset on the actual class labels. For multiclass classifiers, the precision measure is computed either by summing the cumulative true positive and false positives or by averaging the individual class precision over all the classes.

Micro precision represents the classifier and class label agreement by cumulating the per-class results. Macro precision is the average agreement between the classes and the classifier.

In case of MCB predictions, both the numerator ($\mathrm{TP_{class}}$) and the denominator ($\mathrm{TP_{class}} + \mathrm{FP_{class}}$) were zero for tweets in hate (class label 0) and neutral (class label 2) categories, as all cases were predicted to be negative (other class - offensive) for these two classes.

$$Precision_{micro}^{test} = \frac{\sum_{class=0}^{2} TP_{class}}{\sum_{class=0}^{2}(TP_{class}+FP_{class})} = \frac{0+3801+0}{(0+0)+(3801+982)+(0+0)} = 0.79$$

$$Precision_{macro}^{test} = \frac{1}{3}\sum_{class=0}^{2}\left(\frac{TP_{class}}{TP_{class}+FP_{class}}\right)\frac{1}{3}\left(N/A+\frac{3801}{3801+982}+N/A\right) = 0.26$$

4.1.4 Recall

Recall, also called sensitivity, defines a classifiers effectiveness in identifying positive labels. Recall for the hate and neutral categories was zero as none of the classes were identified as positive labels by the MCB classifier. This impacted the macro recall where only non-zero term was for the offensive class.

$$Recall_{micro}^{test} = \frac{\sum_{class=0}^{2} TP_{class}}{\sum_{class=0}^{2}(TP_{class}+FN_{class})} = \frac{0+3801+0}{(0+227)+(3801+0)+(0+722)} = 0.79$$

$$Recall_{macro}^{test} = \frac{1}{3}\sum_{class=0}^{2}\left(\frac{TP_{class}}{TP_{class}+FN_{class}}\right) = \frac{1}{3}\left(\frac{0}{0+227}+\frac{3801}{3801+0}+\frac{0}{0+722}\right) = 0.33$$

4.1.5 F1-Score

F1-score represents the relation between the positive labels of data and classifiers. F1-score is a harmonic mean of performance and recall. Micro F1-score of test predictions was same was micro-and micro-recall.

$$F1 - score_{micro}^{test} = \frac{2 \times Precision_{micro}^{test} \times Recall_{micro}^{test}}{(Precision_{micro}^{test} + Recall_{micro}^{test})} = \frac{2 \times 0.79 \times 0.79}{(0.79 + 0.79)} = 0.79$$

$$F1 - score_{macro}^{test} = \frac{2 \times Precision_{macro}^{test} \times Recall_{macro}^{test}}{(Precision_{macro}^{test} + Recall_{macro}^{test})} = \frac{2 \times 0.26 \times 0.33}{(0.26 + 0.33)} = 0.29$$

4.1.6 AUC

The area under the ROC curve (AUC) represents a model's ability to avoid incorrect classifications. A perfect predictor will have AUC of unity. MCB model classifier had micro-average AUC of 0.50, which was same as making a random guess prediction. Figure 4.3 shows the ROC plot of MCB model predictions on the test data.

$$AUC_{micro}^{test} = \frac{1}{2}\left(\frac{\sum_{class=0}^{2}(TP_{class})}{\sum_{class=0}^{2}(TP_{class} + FN_{class})} + \frac{\sum_{class=0}^{2}(TN_{class})}{\sum_{class=0}^{2}(TN_{class} + FP_{class})}\right)$$

$$AUC_{micro}^{test} = \frac{1}{2}\left(\frac{0 + 3801 + 0}{(0 + 227) + (3801 + 0) + (0 + 755)} + \frac{4556 + 0 + 4028}{(4556 + 0) + (0 + 982) + (4028 + 0)}\right)$$

$$AUC_{micro}^{test} = \frac{1}{2}(0.795 + 0.897) = 0.85$$

$$AUC_{macro}^{test} = \frac{1}{3}\sum_{class=0}^{2}\frac{1}{2}\left(\frac{TP_{class}}{TP_{class} + FN_{class}} + \frac{TN_{class}}{TN_{class} + FP_{class}}\right)$$

$$AUC_{macro}^{test} = \frac{1}{3}\left[\frac{1}{2}\left(\frac{0}{0 + 227} + \frac{4556}{4556 + 0}\right) + \frac{1}{2}\left(\frac{3801}{3801 + 0} + \frac{0}{0 + 982}\right)\frac{1}{2}\left(\frac{0}{0 + 755} + \frac{4028}{4028 + 0}\right)\right]$$

$$AUC_{macro}^{test} = \frac{1}{3}\left[\frac{1}{2} \times 1 + \frac{1}{2} \times 1 + \frac{1}{2} \times 1\right] = 0.5$$

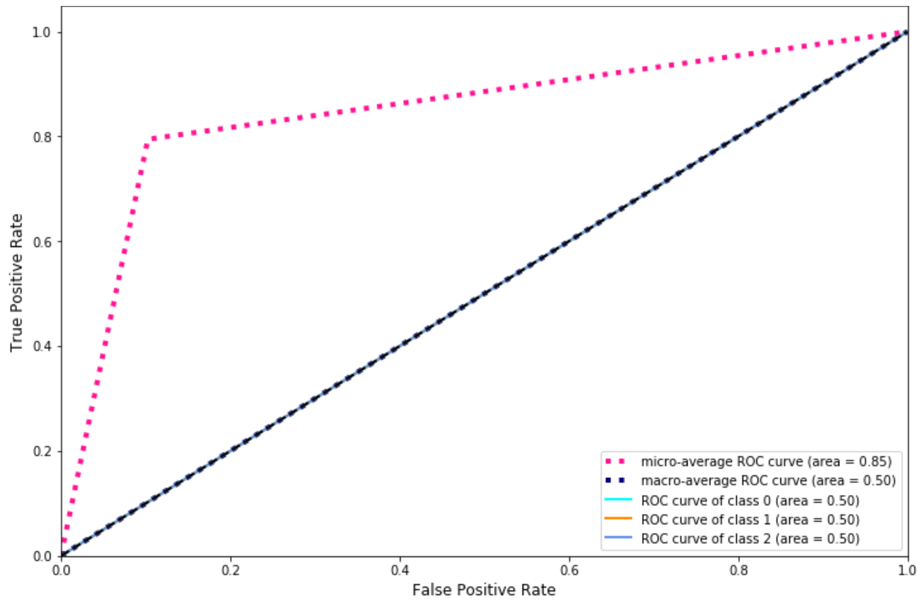


Figure 4.3 ROC curve for MCB classifier

4.2 Random Forest Classifiers

Tree-based models involve stratification of the feature space to simple regions, where an (inverted) tree is grown using splitting rules imposed on the input features at different points (called nodes). Each split-up path (called branch) leads to the final regions (called leaf nodes). Regression and classification problems can be solved using decision tree methods.

Averaging the results of multiple, fully-grown trees on different training data sets could result in reduced variance in the output. Recording more observation or generating synthetic data can help solve the high variance issue. Decision trees rely on a simple, yet powerful technique called *bagging* for obtaining larger training dataset. From a given training dataset of size *n*, *bagging* draws random observations with replacement to generate new data sets of size *n*. On average, the bagged subsets contain 2/3rd of the original data. Trees are then fit on the different bagged datasets, and the majority class is the model's prediction.

Bagging is effective in reducing the variance if the resulting trees are uncorrelated. Stronger features are often selected picked first and most, resulting in similar trees on bagged datasets. Having a large number of correlated trees results in the issue of overfit remaining unresolved. *Random forests* overcome this limitation by randomly selecting a subset of the input features to choose from, at each node. Random selection provides an opportunity for the not-so-strong features to be selected built different trees. Out of say F total predictors, a random subset of f predictors is selected at each node. On average, $(1 - f \,/\, F)$ splits will not even consider the strong features for splitting decision, which results in decorrelated decision trees. Averages from decorrelated trees are less prone to overfitting, making the analysis reliable. Typically, the subset size f is approximately equal to the square root of the total number of features, i.e., $f \approx \sqrt{\sqrt{F}}$.

4.2.1 TF-IDF Vector

The input tweets were first vectorized using the TF-IDF algorithm. Twitter data often contains slangs that are not included in English vocabulary or might be from a different language. Character level feature space can sometimes result in a better classification model [5]. Figure 4.4 shows the distribution of the character length of each word in the vocabulary (training set). The average and median length of vocabulary words were 6. Based on this observation, eight different N-gram features were used to evaluate the random forest classifier:

Character level N-grams where $N \in \{2, 3, 4, 5, 6\}$

1. Word level N-grams where $N \in \{1, 2, 3\}$

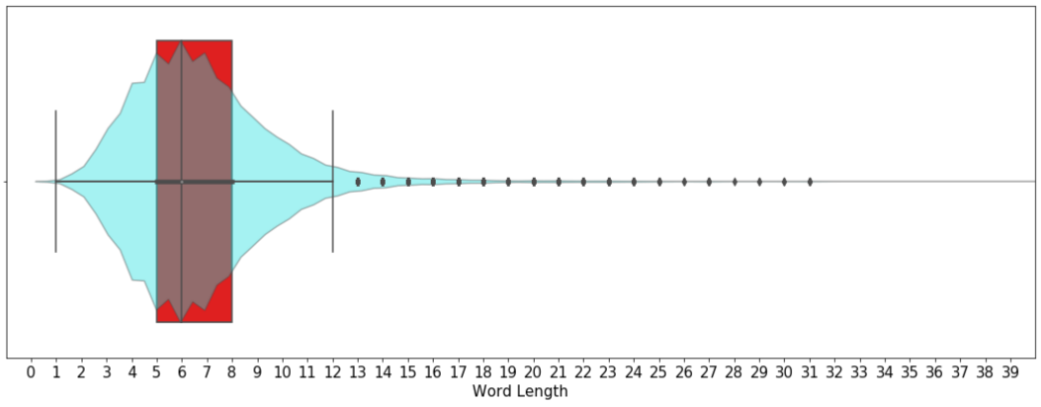


Figure 4.4 Character-length distribution of vocabulary words

4.2.2 Random Forests

The classifier was first tuned using 5-fold cross-validation grid search for two hyperparameters:

1. Maximum number of features per split
   a. $max\_features = \{100,200,500\}$
2. Total number of trees
   a. $n\_estimators = \{500,800,1000\}$

Word unigrams input features were used for the grid search. It was assumed that model performance for other features was the same. The mean validation error (0.83±0.01) and standard deviation (0.12±0.02) did not show any wide variation during the grid search. Hyperparameter values with $max_{features} = 500, n\_estimators$= 100 selected.

Figure 4.5 shows different performance measures for each input feature. The micro measures were more consistent than macro-scores. Also, since the value of precision, recall, and f1-scores (micro-level) was the same per feature, micro F1-score was selected as the performance measures to compare different classifiers.

Table 4.2 and Table 4.3 lists the performance of random forest classifiers character and word level n-grams, respectively. Character level N-gram features performed better than word-level features. Overall performance of character 4-grams was highest while word trigrams had the worst values of the metrics, in line with another study [5]. Word unigrams also performed comparably well. Since the other classifiers (neural networks) were trained using word-level vectorization, word-unigram + Random Forest model was selected as the *baseline model* for this study.

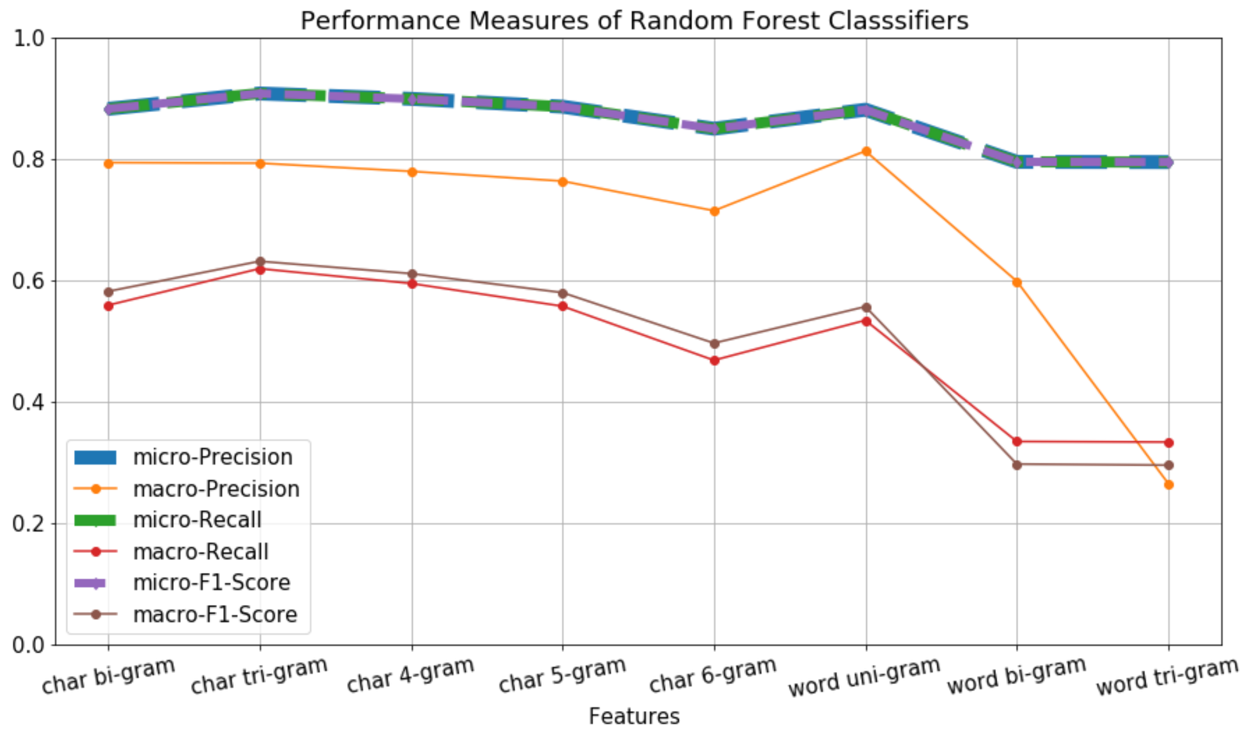


Figure 4.5 Performance measures of random forest classifiers to different features

Table 4.2 Performance measures of random forest classifiers with character N-grams as features

Performance Measures		**Char Bigram**	**Char Trigram**	**Char 4-Gram**	**Char 5-Gram**	**Char 6-Gram**
Micro	**Precision**	0.8844	0.9049	0.9001	0.8850	0.8518
	Recall	0.8844	0.9049	0.9001	0.8850	0.8518
	F1-Score	0.8844	0.9049	0.9001	0.8850	0.8518
Macro	**Precision**	0.8093	0.7840	0.7720	0.7387	0.7622
	Recall	0.5638	0.6130	0.5965	0.5529	0.4728
	F1-Score	0.5847	0.6248	0.6104	0.5752	0.5023

Table 4.3 Performance measures of random forest classifiers with word N-grams as features

Performance Measures		**Word Unigram**	**Word Bigram**	**Word Trigram**
Micro	**Precision**	0.8821	0.7955	0.7949
	Recall	0.8821	0.7955	0.7949
	F1-Score	0.8821	0.7955	0.7949
Macro	**Precision**	0.7586	0.5985	0.5983
	Recall	0.5341	0.3351	0.3338
	F1-Score	0.5551	0.2989	0.2961

Bibliography

[1] "An empirical evaluation of supervised learning in high dimensions," in *25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

[2] C. O. D. Archives, "Hate Speech Identification," 2016. [Online]. Available: https://data.world/crowdflower/hate-speech-identification.

[3] T. Davidson, D. Warmsley, M. Macy and I. Weber, "Automated Hate Speech Detection and the Problem of Offensive Language," in *Proceedings of the 11th International Conference on Web and Social Media (ICWSM)*, Montreal, 2017.

[4] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of ACM,* vol. 18, no. 11, pp. 613-620, 1975.

[5] S. Malmasi and M. Zampieri, "Detecting Hate Speech in Social Media," in *Proceedings of Recent Advances in Natural Language Processing*, Varna, Bulgaria, 2017.

[6] Wikipedia, "n-gram," 02 2011. [Online]. Available: https://en.wikipedia.org/wiki/N-gram. [Accessed July 2019].

[7] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning with Applications in R, Springer Publishing Company, 2014.

[8] T. Whitlock, "Emoji Unicode Tables," 2019.

[9] S. Malmasi and M. Zampieri, "Detecting Hate Speech in Social Media," in *Proceedings of Recent Advances in Natural Language Processing*, Varna, Bulgaria, 2017.