Exploring Ensemble Classifiers and GloVe for Binary Sentiment Classification

Tiffany Huang
Princeton University
twhuang@princeton.edu

Alexis Sursock
Princeton University
asursock@princeton.edu

Abstract

Important social movements such as Black Lives Matter often motivate extreme opinion expression on social media, particularly Twitter. Social media users often prefer to view content that is aligned with their own opinions, which requires content classification. We are interested in identifying classification methods that can accurately categorize Tweets as either pro- or anti-Black Lives Matter. In particular, we employ GloVe (Global Vectors for Word Representation) to evaluate the effect of a recurrent neural network (RNN) on Tweet classification. We find that the RNN, as well as an ensemble classifier composed of other classifiers, tie for the highest accuracy score. We also find that many classifiers experience high false positive rates, which is an aspect to target for future research.

1 Introduction

While the Black Lives Matter movement originated in 2013, the death of George Floyd in Summer 2020 reignited a wave of support for the movement – as well as a fair amount of backlash. Both supporters and opponents of Black Lives Matter turned to Twitter to voice their opinions on the subject. Given that many social media users may want to only view content of one side, and not the other, we are interested in the task of classifying Tweets as either "pro" or "anti" Black Lives Matter, and wish to identify which classifiers would be best for this task. We are also curious about which features can be selected in order to optimizer classifier performance.

Here, we train our classifiers on a data set of 7269 Tweets relating to Black Lives Matter. We evaluate 7 different classification methods, as well as an ensemble classifier composed of 6 of these methods, to determine the best approach to sentiment categorization, using a bag-of-words representation for the Tweets. In particular, we employ the GloVe model (Gloval Vectors for Word Representation) to train a recurrent neural network (RNN). We choose GloVe as our word vectorizer because it enables us to represent the similarity between words as a vector, rather than a scalar, capturing more nuance [**?**]. We evaluate accuracy for each of the classifiers using several metrics to identify the best. We find that the RNN and an ensemble classifier composed of other classifiers, tie for the highest accuracy score. In examining other classifiers, it seems that their lower accuracy was largely due to high false positive rates, or assigning something "positive" that was truly negative.

2 Related Work

We define an ideal classifier as one that not only has a high accuracy score when evaluated on the test set, but also wish to evaluate the F1 score, which will be defined later on in the work. We note that the data set has far more positive Tweets than negative, so we expect a high amount of false positives (classifying something as positive that is actually negative). We expect that the neural network model will perform the best, as one of the benefits of deep learning is that it is able to perform feature engineering by itself [**?**]. Therefore, it can identify and utilize features within the

data set that we as human users may miss, leading to better accuracy. Furthermore, because the GloVe vectors allow for more nuanced calculation of word similarity, we expect that our trained model will perform better with difficult-to-classify inputs. We expect that feature engineering will benefit the classifiers. In terms of feature selection, we believe adding bi grams will lead to better classification; consider how the phrase "Blue Line" likely refers to Thin Blue Line, a pro-police movement, whereas the words "Blue" and "Line" alone do not necessarily have much significance.

Our assumptions are that when using classifiers such as Naive Bayes, that our inputs are conditionally independent. However, in using the neural network classifier, we do not have to make this assumption.

2.1 Data Preprocessing

We initially preprocessed our data set of 8435 labelled tweets by removing hashtags, special characters, and emojis before using WordNet tokenization and lemmatization from the NLTK library. Additionally, this paper solely analyzes the 7269 tweets with either a 'positive' or 'negative' label, allowing for a binary classification problem. Thought was put to the usage of tweets with "neither" labels, but the models' accuracy dropped significantly, forcing us to remove them.

Next, we compared different pre-trained word vector models, including GloVe's Wikipedia 2014 + Gigaword 5 (300d) [1], GloVe's Twitter (200d)[1], and FastText's Wiki-News 2M (300d) [10], for total word coverage on the training set. With 95.2% of all text and 79.7% of all vocabulary found in embeddings[1], the GloVe Wikipedia /Gigaword model proved most fitting and will be used for additional analysis. Removing words which occur only once increases identified portions to 95.7% of text and 85.9% of all unique words. Further, while checking for coverage, slogans, such as BlackLivesMatter, BlueLivesMatter, and AllLivesMatter, names, and written-out emojis composed the majority of unidentified vocabulary. Future research can consider the removal or change in representation of such categories to further improve the model's performance.

2.2 Classification Methods

We use a total of 7 different classification methods, including five from the SciKitLearn Python librararies , one deep learning algorithm trained on GloVe word embeddings, and one Ensemble Classifier combining the others with the exception of RNN.

1. Recurrent Neural Network (RNN): using a 300d vector word embedding from GloVe[1] - we employ this to better capture word similarity and order
2. Support Vector Classification (SVC): using rbf kernel, C = 1.0 - we employ this because support vector machines work well in high dimensional spaces, which is where we operate
3. K-nearest neighbors (KNN): using eighteen neighbors - we choose KNN because it would work well with our data set because of how similar most of the Tweets given are
4. Naive Bayes classifiers (Bernoulli and Multinomial) (NB) - we believe these will add diversity to our ensemble classifier, and they will also be straightforward to employ
5. Random forest (RF): using entropy criterion - we believe RF will be able to handle categorical classification due to its decision tree model
6. Logistic regression (LR): using newton-cg solver - we include this because we viewed logistic regression as easier to optimize initially
7. Ensemble Classifier: combining weighted classifiers 2-6 - we use this to combine the benefits of the above classifiers and also "fight" against high false positive rates

[1]GloVe's Twitter (200d) yielded 92.9%for text and 79.3% for vocabulary, while FastText's model identified 92.3% and 71.9% respectively.

3 Methods

3.1 Spotlight Classifier: Recurrent Neural Networks

A neural network is "a network or circuit of neurons" where neurons are connected to each other; these connections are modeled by weights [2]. In computing, artificial neural networks are composed of several layers of neurons, where each neuron can pass a "signal" (a real number) to those it is connected to [13]. While standard neural networks have a fixed number on inputs and outputs, recurrent neural networks (RNNs) allow for "variable-length sequences as both inputs and outputs" [14]. In addition, RNNs allow for both forward and backward movement between neurons; in the context of sentiment analysis, it means we are able to consider the order of words along with the words themselves in our predictions. Here, we employ a many-to-one model where we take in a large amount of inputs $x_1, ..., x_n$ (these are our input words) and output one output y (a "positive" or "negative label"). RNNs are recurrent because they continue to update hidden states $h_1, ..., h_n$, where each hidden state is calculated by the one prior to it (see Figure 1 in Appendix).

As with all neural networks, the object of the RNN is to minimize loss, which we define through binary cross-entropy loss, which takes into consideration the log probability of an input being positive ($p(y_i)$) through the following equation [6]

$$H_p(q) = (-1/N)\sum_{i=1}^{n}(y_i)(log(p(y_i))) + (1 - y_i)(log(1 - p(y_i)))$$

Here, instead of using stochastic gradient descent to minimize the loss, we use the Adam optimization algorithm, which is an extension of gradient descent that updates network weights from the training data [3]. The process of training can be described as follows (modified from Simeon Kostadinov [9]):

1. Take in input from data set
2. Apply computations to using initially given weights and biases environment
3. Produce a predicted result
4. Compare the result to the expected result (training output) and calculate error
5. "Back propogate" by tracing the same path backwards to adjust variables to reduce error
6. Repeat steps 1-5 until weights and biases are well-defined
7. Apply given weights and biases on an unknown data set (test set)

One issue we run into with RNNs is that they suffer from short-term memory, therefore we employ a Gated Recurrent Unit Network (GRU), which increases memory using mechanisms called gates, or "neural networks that regulate the flow of information flowing through the sequence chain" [12]. Once we train the model on our training set, it can easily be used to predict labels for future samples.

A key assumption of RNNs is that elements must depend on the ones that precede it. It assumes that this is the only dependency, which is not always true as sometimes elements also require information from the elements that follow it [11].

4 Results

4.1 Optimization Strategy

In order to optimize performance, hyperparameter tuning techniques were implemented to find the optimal parameters for individual classifiers. Among others, we utilized Cross Validation and Grid Search to choose the algorithm, weights, and neighbors for KNN, the kernel and C for SVC, and the criterion for RF. As expected, especially classifiers dependent on algorithm selection benefited from hyperparameter tuning with results ranging from an increase of 1.25% from default parameters to up to 2.59% for the RF classifier.

Additionally, we found that tuning the vectorizer and preprocessing choices significantly improved certain classifiers' performance, as described in 2.2. We found that while a combination of unigrams

and bigrams in our Tfidf model improved SVC, it reduced accuracy of Naive Bayes (0.828 to 0.823). We believe this is because NB has strong assumptions about conditional independence of features which, of course, is not provided in a uni- and bigram combination.

4.2 Recurrent Neural Network Performance

After optimizing our choice of word vectors based on coverage, as mentioned in Chapter 2.2, we created an embedding matrix for all identified words and added it to the model. To balance the data set's bias towards positive tweets and reduce false positives we added a balanced class weight parameter. Upon optimizing, the model plateaued with an accuracy of 78.7% and an f1-score of 88.3 after the second epoch, which is usually an indication of the model finding a local minimum for the loss and always predicting the most prevalent label.[7].

We then changed our tweet processing and model fitting technique to better suit binary sentiment analysis[2]. We started by tuning the batch size parameter, which affects both the speed of convergence and generalization power of a model. We identified 128 as the ideal parameter balancing both factors [3]. Using the chosen batch size, several tests were made to determine the right number of epochs to train the model. We found that after 10 epochs any further training would lead to overfitting and reduce the accuracy on the test set. This is evident in the reduced loss rate, but lower test accuracy for 15 and 25 epochs 2. Due to constraints in computing power, we did not run any further optimizations and settled for the 84.32% test set accuracy.[4] Future researchers are encouraged to delve deeper into this model as even a few optimizations and adjustments proved fruitful to increase this model's accuracy by nearly 6%; out of all of our classifiers, we believe this one has the highest upside potential and can even be useful for unsupervised learning projects.

Table 1: A table without vertical lines.

Epochs	1	2	3	4	5	10	15	20	25
Train Accuracy	81.06	81.39	80.64	81.39	81.56	84.64	85.28	87.42	90.12
Loss	49.22	45.80	44.97	42.01	40.77	37.08	33.95	29.97	25.51
Test Accuracy	82.11	82.73	–	–	–	84.32	83.56	–	83.70

4.3 Other Classifiers' Performance

Of the other classifiers we explored, we found that the Support Vector Classification and Logistic regression classifiers performed the best. We drew the ROC curves (receiver operating characteristic curve) for each classifier (see Figure 2 in the appendix). Here, we plot the true positive rate and the false positive rate, where we define $TP = \frac{TP}{TP+FN}$ and $FP = \frac{FP}{FP+TN}$ [4]. The figure also displays the AUC (area under the curve) score for each classifier; a score closer to one indicates better performance. We also calculate the precision, recall, and F1 score for each of these classifiers. We define $recall = \frac{truepositives}{truepositives + falsenegatives}$ and $precision = \frac{truepositives}{truepositives + falsepositives}$. We define the F1 score to be $F1 = \frac{2 * precision * recall}{precision + recall}$.We do so by performing 5-fold cross validation, calculating each of these scores, and calculating the average. The results:

We can see that Logistic Regression and SVC both perform fairly well, and have the highest F1 scores. Interestingly, we note that Random Forest has a recall score of 1 yet a somewhat low preci-

[2]The two different techniques used can both be found in the attached notebook. The latter, improved model is found under "RRN Model 1".

[3]Other batch size values tested were 64, 256, and 512. While 64 provided a less reliable accuracy and converged to a local extremum, the larger values were slower at converging to higher accuracies and were thus too computationally heavy.

[4]With the help of Princeton Research Computing, three 10-core servers were used to run this model. Even then, the fitting of a single epoch took us between 300 and 400s each.

Classifier	Precision	Recall	F1 Score
Random Forrest	0.8139	1	0.8974
Logistic Regression	0.8393	0.9838	0.9058
SVC	0.8541	0.9684	0.9078
KNeighbors	0.8540	0.9324	0.8915
Multinomial NB	0.8192	0.9963	0.8991
Bernoulli NB	0.8555	0.9165	0.8850

Table 2: A comparison of evaluation metrics for selected classifiers (table format source: [5]

sion rate. This seems to indicate that there are a high amount of false positives; this is true for all of the classifiers as precision is greater than recall for all. This confirms our expectation that there would be many false positives due to the imbalanced nature of the training set.

A clarifying note: the above table was calculated when we did not perform feature engineering (meaning we did not add bi grams or max features to the data features). This is because when calculating the evaluation metrics for these classifiers with feature engineering, we found that the results were the same or worse. Therefore, we conclude that feature engineering did not benefit these classifiers substantially.

4.4 Ensemble Performance

Given our base classifiers' diverse range, we decided to combine the weighted prediction of them to form an ensemble classifier to improve the overall accuracy. As the data set's distribution of positive and negative tweets favors the former and as individual classifiers' show a high rate of false positives, we summed all classifiers' predictions using the following formula:

$$Positive : +weight * weight$$
$$Negative : -2 * weight$$

This adjustment puts a higher weight on the less likely "negative" prediction and puts a higher emphasis on the model's individual accuracy, which is equated to it's weight ([0;1]). The ensemble yields a test set accuracy of 84.32%, nearly 0.4% higher than the highest individual base classifier and equivalent to our optimized Recurrent Neural Network classifier.

5 Discussion and Conclusion

In this paper, we optimized and analyzed seven classifiers' predictive power for sentiment analysis of labelled tweets on the BLM movement. We combined a wide array of models, including two versions of a deep learning neural network and an ensemble classifier. We found that, on accuracy, both the ensemble classifier and the RNN proved superior to the other individual classifiers with an accuracy level of 84.32% on the testing set. In our other classifiers, we saw relatively high F1 scores, though they did suffer from high false positive rates.

For future researchers, we believe that the neural network is best suited for additional optimization. Given its computational heaviness, we were unable to tune it to the fullest extend possible and additional optimizations such as preprocessing for slogans and names, and experimentation with different algorithms, methods, and parameters, as identified in this paper, are possible. A further extension we recommend is the addition of the RNN model to the ensemble classifier, which could not be achieved due to lack of time for exploration.

In summary, we believe that both models proved valuable for Twitter sentiment analysis and should be studied further to improve their accuracy and computational viability.

Acknowledgments

We want to thank Dr. Li for guiding us to Princeton Research Computing to run our RNN code on the Adroit Server. We also want to acknowledge Siena Dumas Ang, Deniz Oktay, Yaniv Ovadia, and Sulin Liu for helping us in Office Hours, on Ed, and via Email.

References

[1]

[2] Neural networks. https://en.wikipedia.org/wiki/Neural_network.

[3] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam.

[4] Google Developers. Classification: Roc curve and auc. ="https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[5] Barbara E Engelhardt. Fast classification of newsgroup posts. 2021.

[6] Daniel Godoy. Understanding binary cross-entropy / log loss: a visual explanation. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[7] Jeremy Jordan. Setting the learning rate of your neural network. https://www.jeremyjordan.me/nn-learning-rate/, 2018.

[8] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. "https://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[9] Simeon Kostadinov. How recurrent neural networks work. https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7.

[10] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[11] peteris. Why use a recurrent neural network over a feedforward neural network for sequence prediction? https://ai.stackexchange.com/questions/16226/why-use-a-recurrent-neural-network-over-a-feedforward-neural-network-for-sequenc, 2019.

[12] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21, 2018.

[13] Wikipedia. Artificial neural networks. ="https://en.wikipedia.org/wiki/Artificial_neural_network.

[14] Victor Zhou. An introduction to recurrent neural networks for beginners. https://victorzhou.com/blog/intro-to-rnns/, 2019.

6 Appendix

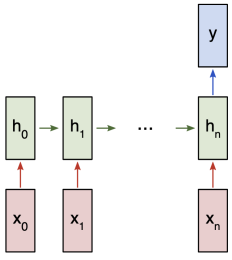

Figure 1: A many-to-one RNN, courtesy of Victor Zhou [14] and Andrej Karpathy [8]

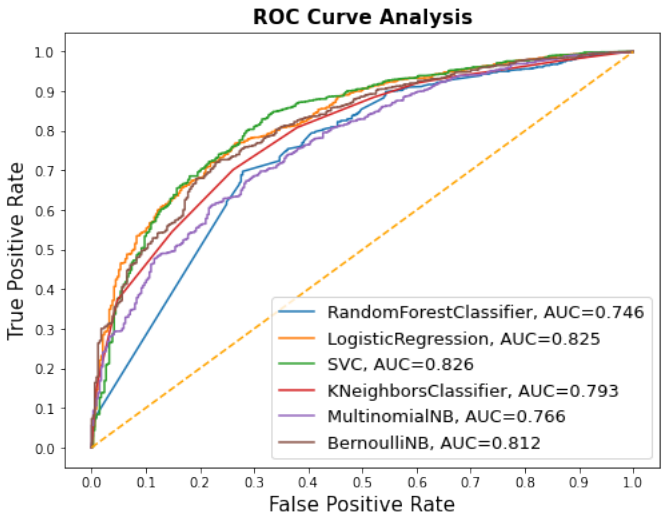

Figure 2: ROC Curves of each classifier independently (RNN and ensemble classifier not included