Classifying Twitter Sentiments from the Black Lives Matter Movement

Hirsh Guha
Princeton University
hguha@princeton.edu

Josh Cohen
Princeton University
jmc16@princeton.edu

Abstract

We classify tweets related to the Black Lives Matter movement as positive or negative in order to ascertain an understanding of various classifiers. After cleaning the data, we train 5 different classifiers, each using both a bag-of-words representation and a tf-idf score. We use a variety of metrics to evaluate the performance of the classifiers, including accuracy, precision, recall, ROC AUC scores, and cross-validation. We found that the classifiers generally had better performance with tf-idf scores, and that Support Vector Machines, Naive Bayes, and Logistic Regression tended to be the most accurate by a variety of metrics, while Random Forests and K-Nearest Neighbors performed poorly.

1 Introduction

In a world where opinions spread like wildfire on social media, sentiment analysis can be a powerful tool to quickly gain real-time information about people's opinions on brands, policies, or social movements. This social media data is unfiltered and can be widely distributed, allowing quicker and possibly more accurate information than traditional polls or other opinion-tracking measures. In summer 2020, the Black Lives Matter movement provoked a range of reactions on social media. We would like to classify those reactions into positive and negative categories in order to better gauge sentiment on this or similar social movements in the future. To do so, we will examine the performance of a variety of classifiers, both generative and descriptive, on a dataset containing roughly 8500 tweets. We analyze the results with a variety of metrics in order to effectively compare the classifiers with one another for this task.

2 Related Work

Kouloumpis et al. investigated the utility of existing lexical resources and features that capture information about the particular style of speaking popular in microblogging [2]. Since the advent of Twitter has born a sublanguage unto itself, they seek to analyze whether there is useful information that can be parsed from the vernacular. While their methods are intrinsically different than ours (They use AdaBoost), and their dataset is not particularly focused on a subset of tweets, our goals in creating predictive results for twitter sentiments are very similar.

2.1 Expected Behaviour

The tweets in the dataset have several important features that will affect the performances of the chosen classifiers, and which allow us to make some predictions. First, there are roughly 4000 positive tweets compared with roughly 1000 negative tweets in the training data. This means that approaches such as k-nearest neighbors may not perform as well, since there are many more potential positive neighbors than negative ones. Secondly, the language used in the tweets is often very similar. In particular, many positive and negative tweets use the words "black", "lives", and "matter",

and contain mentions of cops or police. On first glance, it seems like it may be harder to see a distinction in many of the tweets as opposed to, say, reviews, where certain words like “love”, or “disappointed” are highly predictive.
We would expect this similarity between many of the tweets to have several consequences. First, the data may not be linearly separable, which could cause Naive Bayes and logistic regression to perform poorly. Additionally, if the datapoints are relatively close to another and there is not as much clustering, k-nearest neighbors may again perform relatively poorly. Finally, since there are some words that seem to be very discriminatory between the categories (for instance #alllivesmatter appears almost exclusively in negative tweets), we may be able to improve performance by using tf-idf scores to take frequency into account.

2.2 Dataset

The dataset consists of 6747 training tweets and 1688 test tweets, all labeled as “positive”, “negative”, or (in the training data) “neither”.

2.2.1 Feature Selection

In order to clean the data and build our models, we use a config that allows us to toggle each of the following steps:

1. Remove words containing numbers (these words generally appear only once and tend to skew the most predictive features).
2. Turn emojis into traditional ASCII characters using the python emoji package.
3. Remove the word “RT”, URLs, and user mentions from the text of the tweet.
4. Remove capitalization and punctuation from each tweet.
5. Use the NLTK python package to remove stopwords (ie, “a”, “the”, etc) and to lemmatize the words, or to decompose them into their base words (ie, talked $\rightarrow$ talk). We need to remove “all” from the list of stopwords, since that word is particularly important for this project.
6. Finally, we remove the tweets labeled “neither” in order to more clearly focus on the difference between positive and negative tweets.

We can then use scikit’s χ^2 feature selection tool to choose various values of the k best features with which to use in the training and test data. It requires some testing to find a value of k that suitably increases accuracy and improves speed without losing much needed information. We settled on $k = 500$, which is roughly 4% of the total features.

Finally, we used a Vectorizer to turn the data into a sparse matrix. We used both a CountVectorizer (in which a_ij is the number of times that word j appears in tweet i) as well as a TfidfVectorizer (which weights words by how frequently they appear in a category and inversely by how common they are in all categories). In each case, we remove the bottom 0.1% of words by each metric, as we found that without this step, the most predictive features tended to be words that appeared in a single tweet and thus had a very high correlation with one category.

3 Methods

We selected 5 classifiers from the scikit-learn library in order to compare across models [5]. The classifiers were as follows:

- *Multinomial Naive Bayes (NB)* - Simple, fast, and generative. It is a useful benchmark to see if more sophisticated and complex models would perform significantly better.
- *Logistic Regression(LR)* - Simple, fast, and discriminative. In theory, it should have a smaller error than Naive Bayes with a large enough sample size[3]. This serves as a useful benchmark and a way to see if the independence assumptions of the model hold true.
- *K-Nearest Neighbors(KN)* - This classifier requires no assumptions about the data. We want to see if our prediction that this classifier would fare poorly due to the structure of

the data holds. It needs to be run multiple times to find a good value for k, the number of neighbors to match.

- *Random Forest(RF)* - This classifier makes no assumptions of the input data and generally performs better than logistic regression about 70% of the time[1]. We wanted to see if this result holds true, particularly if the data did not turn out to be linearly separable or have independence between features.
- *Support Vector Machines(SVC)* - This classifier attempts to find a separating hyperplane that maximizes the separation. SVM has been found to outperform Naive Bayes on sentiment-analysis related tasks[4]. We wanted to test this result as well.

We have 2 sets of results for each of these models: using a bag-of-words approach and using tf-idf scores. All the models are identical in each case, except that we changed the α smoothing parameter for naive Bayes from 1 to 0.1 for tf-idf so that it would not overwhelm the data.

3.1 One Model In Depth - K-Nearest Neighbors

K-Nearest Neighbors is a very simple classifier based around the idea of classifying points in some space based on their distance to the points in the training set [6]. To be more precise, suppose that our data comes from some space S (for instance, in this project, we can think of this space as $\mathbb{R}^{|vocab|}$). Additionally, we have some distance function $d : S \to S \to \mathbb{R}$.

Given a feature vector $D = \{(x_1, z_i), (x_2, z_2), ..., (x_n, z_n)\}$, where the $x_i \in S$ are the samples in the training data and the $z_i \in \{0, 1\}$ are the corresponding classifications, we do the following:

- For a given sample x^*, find the K samples from $x_1, .., x_n$ such that $d(x_i, x^*)$ is minimized.
- Let $Z = \{z_1, ..., z_k\}$ be the labels of the x_i's found. Assign $\hat{z}^*$ to be the majority class in Z.

This classifier requires a value for K and a distance function d. A good choice of K may depend on the size and the structure of the data, while a choice for d may depend both on the structure of the data and on the space itself. Common choices include Manhattan distance, Euclidean distance, or the generalization: Minkowski distance, each of which are given, respectively, below:

$$d_{Man}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \qquad d_E(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad d_{Min}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

The implementation in scikit-learn, which we used, uses Minkowski distance by default, but also by default chooses $p = 2$, making it equivalent to Euclidean distance.

As a classifier, K-Nearest Neighbors has a number of notable features compared with its peers. First, it requires no training, since it simply compares the input to the training data points. It also makes no assumptions about the structure of the data or about the independence of various features. It can easily be extended to handle multiple classifications (by choosing a plurality instead of a majority in Step 2), and it is kernalizable - ie, the data can be first mapped to a higher dimensional space by a kernel function κ, and then we can run K-Nearest Neighbors on the higher dimensional space. Its (testing) runtime is $O(np)$ where n is the number of samples and p is the number of features.

However, the simplicity of this approach leads to some drawbacks. First, a relatively small number of outliers or skewed data can cause problems, as they may play an outsized role in the distances chosen in step 1 (particularly with small k). In general, choosing an appropriate k and distance function d may not be easy, and the wrong distance function d could easily lead to irrelevant features playing a massive role. For example, if we have features $x \in [0, 1]$ and $y \in [0, 100]$, using Euclidean distance would cause x to be almost irrelevant in the calculation. Additionally, if the data is very skewed, more distances may come from one of the classes purely through sheer numbers (consider a training set skewed 90-10, for instance). This can be resolved by weighting the distances by the prevalence of the classes. More generally, the model does not give probabilities and instead just outputs a class,

Name	Accuracy	Precision	Recall	CV Accuracy	CV Stddev	ROC AUC Score	Time (s)
NB	0.836	0.884	0.919	0.840	0.023	0.840	0.3
LR	0.841	0.855	0.965	0.848	0.018	0.837	3.3
KNN ($k = 50$)	0.815	0.826	0.972	0.821	0.020	0.725	4.5
RF	0.826	0.857	0.924	0.825	0.023	0.775	27.5
SVC	0.839	0.846	0.973	0.847	0.019	0.838	6.9

Table 1: Results with CountVectorizer

Name	Accuracy	Precision	Recall	CV Accuracy	CV Stddev	ROC AUC Score	Time
NB	0.829	0.858	0.988	0.843	0.014	0.863	0.1
LR	0.835	0.861	0.978	0.845	0.007	0.855	2.2
KNN ($k = 50$)	0.817	0.835	0.996	0.822	0.002	0.754	3.6
RF	0.827	0.868	0.933	0.826	0.025	0.795	33.2
SVC	0.844	0.877	0.962	0.851	0.018	0.860	1.6

Table 2: Results with TfidfVectorizer

which may mask significant uncertainty and provides less useful information. Thus, K-Nearest Neighbors can best be classified as a simple, general-purpose approach that can be effective with well-chosen k and d values, but does require some care in these choices.

4 Extensions

In addition to training and comparing 5 classifiers, we chose to extend the project in two different recommended ways to further our exploration of twitter sentimental analysis. First, we used a variety of feature selection methods, including χ^2 feature selection and pruning many words that were much less likely to be predictive (for instance, those with numbers). Second, we extend the given vectorization methods to compare a traditional CountVectorizer against a TF-IDF Vectorizer, and show results for both, enabling comparison among both the classifiers and the data representation.

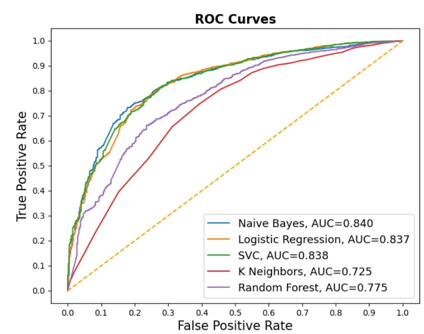

Figure 1: ROC Curves for Bag-of-Words

5 Results

Tables 1 and 2 summarize several metrics for each vectorizer and classifier. Additionally, we plotted the Receiver Operator Characteristic (ROC) curves for each method. This gives a visual method of determining how effective each model is at distinguishing true positives from false positives at each probability threshold. These figures are shown on the next two pages. Note that we partitioned the data into testing and training samples randomly; since there were far fewer negative tweets in the dataset, there are few false negatives, so some of the recall values are quite high. Finally, Table 3 shows the five most predictive words for each category for the Naive Bayes model (scikit-learn does not provide this capability for all models, and Naive Bayes performed well in many of the above metrics).

6 Discussion

The above tables show that many classifiers improved in precision, recall, CV accuracy, CV stddev, and AUC score when using tf-idf data rather than bag-of-words. This is unsurprising, as we would expect that the relative frequency of words in each category plays a large role in prediction. As noted in Section 2, some very common words may be less predictive (indeed, Table 3 does not list

Negative (count)	Positive (count)	Negative (tf-idf)	Positive (tf-idf)
answer	antoniomartin	break	another
everyone	animal	anonymous	anger
hope	decision	everyone	decide
ally	activist	berniesanders	act
bill	destroy	ca	deserve

Table 3: Most Predictive Words with Naive Bayes

some expected words such as "#blacklivesmatter", likely because they are used by both positive and negative tweets). Our results also show the importance of cross-validation. The CV accuracy was usually better than the accuracy over the whole dataset, which illustrates that we may have been "unlucky" with the whole dataset, and more trials on different data would increase accuracy.

We found significant differences in accuracy and ROC AUC score for the chosen classifiers. NB, LR, and SVM were very close in performance, while RF and KNN were less accurate. On the other hand, RF was comparable to the above classifiers in precision, yet less accurate and with a significantly lower recall. In most of these metrics, there was no significant relative change between bag-of-words and tf-idf, as the same models did the same amount better in each case. This may suggest that the models' performance was based much more on the underlying structure of the dataset relative to how such structure compared with the model assumptions, rather than the word frequencies or counts. However, Naive Bayes had the best AUC score, SVM and LR had higher accuracies, and the results for precision and recall were split, suggesting that the best choice of classifier heavily depends on which metrics are being optimized. We note that the accuracy for all the models, save RF, increased with χ^2 feature selection (we do not show the original data due to space constraints). This may be because RF is better at finding discriminating features in the data even among features with dependencies. Lastly, we note that the top three models were about 84%-85% accurate on the cross-validated data, and none of these models were able to dramatically outperform the others. This may speak to the difficulty of classifying tweets that often use similar language, and may require significant context in order to interpret them effectively.

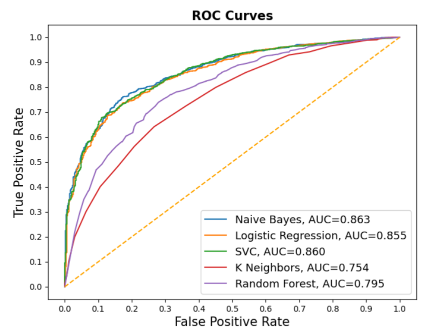

Figure 2: ROC Curves for TF-IDF

7 Conclusions & Future Work

We analyzed roughly 8500 tweets about the Black Lives Matter movement to determine which classifiers would perform best in determining if tweets expressed positive or negative sentiment. We performed trials using a both bag-of-words approach and a tf-idf approach. As predicted, using tf-idf scores words generally improved performance, but there were not huge differences. In general, Support Vector Machines, Naive Bayes, and Logistic Regression performed the best, while Naive Bayes and K-Nearest Neighbors were the least accurate. The top models achieved about 84%-85% accuracy in cross-validation.

There are two main directions in which to extend this work. First, we can use more sophisticated or specialized models rather than the general-purpose models implemented in the scikit-learn library to determine if the $\approx 85\%$ accuracy can be improved or if this is intrinsic to the data. Second, we would extend this analysis to datasets of other tweets to determine if the features of this dataset are unique to the discourse around the polarizing Black Lives Matter movement or if this sentiment analysis is similar to other applications.

References

[1] Raphael Couronné, Philipp Probst, and Anne-Laure Boulesteix. Random forest versus logistic regression: A large-scale benchmark experiment. *BMC Bioinformatics*, 19, 07 2018.

[2] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis: The good the bad and the omg! *Proceedings of the International AAAI Conference on Web and Social Media*, 5(1), Jul. 2011.

[3] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 841–848, Cambridge, MA, USA, 2001. MIT Press.

[4] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, page 79–86, USA, 2002. Association for Computational Linguistics.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.