Stat 154 Final Project: Twitter Sentiment Analysis

Dashiell Stander

May 4, 2016

1 Introduction

Sentiment analysis is one of the oldest problems in Natural Language Processing (NLP). It has great value both as a purely academic goal as well as numerous applications in the private sector. For this project, we were given more than 1.5 million tweets, 140 character strings posted to the social media site Twitter, and tasked with classifying them as expressing a "positive" or "negative" sentiment, coded as a 1 or a 0, respectively.

2 Feature Engineering

The data for this project was raw HTML source code, so there was quite a lot of room for new features to be engineered. Indeed, it quickly became apparent that the problem would respond more effectively to clever feature engineering than either model choice or clever thorough parameter tuning.

The first step was to make the data readable to the tokenizer. For example, all of the '<3' and ':-*' instances were changed to 'HEART' and 'HAPPYEMOJI'. This allowed me to use scikit-learn's off-the-shelf tokenizer while still aggregating the data in such a way that the model would know that ":-)" and ":)" are expressing the same sentiment. A partial list of tokens that were aggregated is as follows:

- All "happy emojis" with smiling, silly, or kissy faces were replaced as "happyemoji".
- All "sad emojis" with frowning, dismayed, or grimacing faces were replaced as "sademoji".
- All "@twitter-handle" style mentions were replaced with "@MENTION".

The second step was to differentiate between words and their semantic negations. "I like you" and "I don't like you" are expressing the exact opposite sentiment, despite their similarity according to a naive bag of words model. As such, I "tagged" each token between a negation (i.e. "can't", "won't", "not", etc...) and either the end of the phrase or the tweet, whichever came first, as "not[token]".

After the raw data had been modified in this way, I used sklearns TfidfVectorizer to create a Tfidf-weighted Document-Term-Matrix, selecting the 10,000 bigrams with the highest Tfidf scores. This transformed data was what I used to train and test the models.

3 Models

3.1 Context

I tried many models, and ruled out some relatively quickly (kernel SVM because it was too slow) and others later on (linear SVM because it was slightly less effective than Logistic Regression and gave otherwise very similar results). What follows are the details of the two most succesful models. With the exception of XGBoost, all models were the scikit-learn implementations.

3.2 Gradient Boosting

Many models were quickly weeded out due to how slowly they dealt with the massive 1,528,627 x 10,000 matrix that made up the final dataset, but gradient boosting with the XGBoost package dealt with it extremely well and provided a lot of flexibility. To limit the feature space when tuning the hyperparameters, I focused on tuning the learning rate "eta", the regularization parameter "gamma", the subsample amounts (columns and rows), and the number of boosted trees. I paid special attention to make use of the "early_stopping_rounds" feature so that more trees than strictly necessary were never used.

3.3 L2 Logistic Regression

Logistic regression was also very consistently fast and also had fewer hyperparameters to optimize, which made it an attractive option. I looked at the L1 vs. L2 penalty, whether or not to use a weighted prior in training, and the regularization parameter C.

4 Results

4.1 Model Results

All the models were tested with three-fold cross validation. The other models that I tried performed similarly, if not quite as well. One interesting thing

not reflected in the following table was that the subsample amounts controlled overfitting much more effectively than gamma.

Model	Parameters	Error	AUC
XGB	eta: .65, gamma: 5, row subsample: .75, column subsample: .75	.2274	.827
Logistic Regression	penalty: L2, C: .8	.2179	.7836

The discrepancy between Error and AUC is interesting given how closely balanced the classes are in the sample.

4.2 Unsuccesful Models

In some ways, I learned more from two ideas I had that were unsuccessful– an ensemble voting classifier and a hierarchical model. The ensemble voting classifier used XGBoost, Random Forest, Naive Bayes, Logistic Regression, and Linear SVM in hopes of averaging out the error in all of them.

The hierarchical model I implemented after realizing that gradient boosting and Logistic Regression were the most effective models thus far. I had XGBoost pass through the training data and then split up the dataset into those that XGBoost classified as 1 and those it classified as 0 (each unbalanced, with about 77% actually being ones or zeros). Then I fit a logistic regression model to each one, given the prior of the unbalanced classes.

Neither of these models, however, worked at all better than the simple gradient boosting or logistic regression models. In fact, the hierarchical model performed quite a bit worse (around 66% accuracy).

5 Conclusion

There are two big takeaways from this project: feature engineering works and some problems can be quite intractable.

Despite extensive computer time spent tuning hyperparameters, it never made a difference of more than one percentage point. The larger gains came from using feature engineering to emphasize the truly information-rich features of the data.

I say that some problems are intractable is because, while I had some hope that there were some features in the dataset that, if I were just clever enough, I could get my models to home in on. This turned out not to be the case. That the ensemble classifier showed no improvement over the best of the single models indicates that each model was misclassifying, more or less, the same tweets. There was no gain to be had from averaging because all the averages were about the same. Similarly, I thought that there might be some unifying characteristics of the tweets that got misclassified that logistic regression may have been able

to pick up on if it was focused on just those datapoints. This also, however, proved not to be true. Improvements could be made on telling the difference between correctly classified and misclassified data points in the training set, but this proved to entirely come from overfitting and only worsened performance on the test set. This indicates that there were no systematic features to the tweets that were misclassified, only noise to be found in individual training sets.

Which is not to say that this problem and the correct classification of those tweets is entirely intractable, but that I could not do it within the scope of this project. Further explorations into following Hoang Duong's example and using the results of different models as the inputs into a neural net would be my first step if I were to continue this project. The second would be to use more sophisticated feature selection–perhaps inititally choosing 20,000 or 30,000 features and then using correlation of F1 score to narrow them down instead of just raw Tfidf.