Team Ball is Life - CS 155 Sentiment Analysis

By Rushikesh Joshi and KC Emezie

Overview

The purpose of this project was to produce a highly accurate model that would predict the sentiment of a speech (positive or negative) by utilizing supervised training on a set of labeled speeches, and then applying our model to a test set of speeches that were not labeled. As part of the competition we were provided with a bag of words that contained the top 1000 most common words of which the speeches consisted of. This bag of words representation was then correlated to each speech by a frequency count, denoting how many times each of those words appeared in each speech. The words provided in the bag of words were stemmed, and had stop words omitted. As mentioned earlier, we were given, as training data, the frequency distribution of the words for 4189 speeches along with the sentiments for each of the 4189 speeches. Our objective was to then predict the sentiments for a test data set consisting of the word frequencies for 1355 speeches.

In order to optimize our score in the competition, the members of this project employed a strategy that consisted of trial and error supported by research analysis. After an initial implementation to gauge our standings, our approach was divided into two fronts:

1. Research of normalization techniques by Rushikesh in order to discover data manipulation techniques that could potentially increase our classification accuracy.
2. Research of classifier combination methods in order to leverage the accuracies of various classifiers into one consolidated prediction.

Throughout our research, both members were also on the lookout for new classifiers that would better fit the dataset we were training and predicting on.

One of the key challenges of the project was the inconsistency of model performances when cross validating/training versus modeling the actual test data. This will be discussed in greater detail in following sections, but throughout the competition we found it difficult to fine tune models and select classifiers because accurate predictions on split training data or during cross validation did not necessarily correlate with a better modelling of the actual test data. Although we were not sure if this was due to the test data having a different distribution than that of the training dataset or merely due to errors on our part, we were able to gradually improve our score by adapting our strategy to successful submissions on the leaderboard.

Data Manipulation

For this project, the only data manipulation that was conducted was a normalization of the bag of words frequency counts given for each speech. Prior to honing in on a successful predictive model, we did some research online to better understand bag-of-words problems as well as sentiment classification problems. Our

first pass at creating a predictive model was using a Decision Tree Classifier, because we just wanted to get points up on the scoreboard. However, after reading more about bag of words problems, we learned that linear classifiers are traditionally better suited for sparse, high-dimensional data. Since we have only 1000 features, it was really unclear if this was considered “high enough” with respect to dimensionality, because we actually saw very poor performance with linear classifiers like logistic regression. Thus we could have tried to implement more features for our models, but we didn’t have enough time to formalize this idea and implement it. Some articles we read also said that weighting should be done for words that tend to appear more frequently, or are more common. The normalization method that we decided on was Term Frequency-Inverse Document Frequency (tf-idf). Tf-idf is a very common method for emphasizing words that appear frequently in a given document (or speech in our case), and de-emphasizing extremely common words when considering the entire data set (all the speeches in our case). The metric for term frequency was based on an augmented frequency to prevent a bias towards lengthier documents

Term frequency = 0.5 + 0.5 * (word_freq / max_word_freq_in_speech)

Inverse Document Frequency = log(N / num_speeches_with_word)

The final tf-idf metric was then a product of the above two metrics. In our first pass, we implemented this normalization, but after noticing odd decreases in our accuracy scores due to the idf, we removed the idf portion of the metric, and decided to normalize our data on a speech-by-speech basis. Thus, in our final normalization, each word count in

the frequency distribution for the bag of words for a given speech was modified to reflect:

Term frequency = 0.5 + 0.5 * (word_freq / max_word_freq_in_speech)

This gave us an improvement in our accuracy scores, and seemed to make sense to us because we were not as concerned with how common a word was across all speeches, since we were ultimately interested in a binary classification of one speech at a time. One final idea that we learned by researching online, was that the omission of stop words could reduce the accuracy of models. Since we did not have the original data, we could not re-create the bag-of-words by adding stop words, but this would have been a very interesting approach to follow if the necessary data were available.

Learning Algorithm

In order to improve our model's accuracy we experimented with a number of different classifiers detailed below. We initially started with an implementation of the decision tree classifier as we had experience working with this classifier in a previous set. We then, through basic research and the use of the python sklearn package, experimented with a number of different classifiers and ended up constructing a fairly accurate model that consisted of combining the predictions of our most successful submissions. The oddest part of our experience was that we saw extremely low correlation between the cross-validation scores achieved on our training set, and actual performance on the test set. This confused us, and seriously hindered our ability to select models based strictly on performance, since we could only reliably use models

that we had submitted for evaluation. This in turn was hindered by the 5 submissions-per-day limitation enforced by Kaggle. To elaborate this a little further, when improvements in accuracy were observed using sk-learn's cross-validation implementation and on our self-created validation set, these did not always (in fact very rarely) correspond to accuracy score increases on the leaderboard. Thus, when choosing models for our final combination, we judged the "ability" of a model not based on the cross-validation score we observed, but rather on the model's performance on the test set (via Leaderboard). The starkest example of one such discrepancy was logistic regression, which was recommended by online articles. We saw ~74% accuracy using logistic regression models, but when we submitted our best logistic regression model to Kaggle, we achieved ~58% accuracy on public score, and ~61% on private score.

Key Algorithms:

1. **Decision Tree [min_samples_leaf=15]: (Public score: .61834, Private score: .61561)**

   Python sklearn implementation**.**
2. **SVM [Default Parameters]: (Public score: .61243, Private score: .63770)**

   Python sklearn implementation**.**
3. **Naive Bayes [Default Parameters]: (Public score: .53550, Private score: .56406)**

   Python sklearn implementation**.**

4. **GBC [Default Parameters]: (Public score: .64497, Private score: .67747)**

   Python sklearn implementation**.**

5. **Logistic Regression [Default Parameters]: (Public score: .57988, Private score: .61708)**

   Python sklearn implementation**.**

6. **Bagging Classifier [Default Parameters]: (Public score: .63166, Private score: .67010)**

   Python sklearn implementation**.**

7. **Combination of top submissions: (Public score: .64941, Private score: .68041)**

   Original algorithm. Leveraged shared predictions across our top submissions on the leaderboard.

Model Selection

1) Train all of the different types of models with default parameters on word frequency.
2) Check the performance of each model using cross validation. Models that significantly underperformed with the defaults were ignored from here on out.
3) Check performance of each model using a training test split. We split the training data into ⅔ training, ⅓ testing in order to resemble the competition and then analyzed accuracy of model predictions in order to determine which models were worth submitting to kaggle.

4) Slight parameter adjustment was done based on cross validation and training split accuracies but as mentioned above improvements did not necessarily correlate with higher accuracy on test data.
5) Model predictions were submitted for scoring, and the top five performing classifiers were chosen for our combination model.
6) Analyzing the models and their respective accuracies, we figured that we could maximize the number of correct predictions by determining which predictions were shared across the most accurate models. We then built a model which checked to see if a prediction was shared across all our top submissions and if so used that prediction. In the case a prediction was not unanimous a simple "vote" was cast between the models and the most frequent prediction was used. This submission ended up being our most accurate algorithm.

Conclusion

Upon completion of the competition, our team finished in 31st place out of 56 teams; not a "true" 31st place however, as multiple teams tied with the same scores, but were ranked in order of earliest submission. While not satisfied with our final result, we were consoled by the fact that improvements above our classification algorithm were very minimal, until the top two places. Our final model that utilized multiple predictive models through a voting based combination performed fairly well on the test set, given much lower scores on the public leaderboard. Thus, we were happy with our ascent from public to private, as we tried to keep our models simple enough that overfitting

would be minimized when we transitioned to the private leaderboard. While we initially used trial-and-error to experiment with various classifiers, once we started to identify classifiers that performed best with default parameters, we then adopted an iterative approach to test for optimal parameters using cross-validation via sk-learn's implementation, as well as accuracy scores based on our ⅔ and ⅓ partitioning of the training data to create a validation set. While we were happy with the performance of our combinatorial model approach that gave us our highest accuracy score, if we had more time we would have definitely implemented a more robust ensemble selection model. We researched ensemble selection and generalized classifier techniques, but due to time limitations with other projects and problem sets, we unfortunately could not follow these ideas all the way to completion. Given more time, we definitely would have used an ensemble selection method to utilize many more predictive models, to help generate a more predictive final model.