tb.lx Coding Challenge Report

Sean Fitzpatrick

12/23/2019

**Introduction**

I had never heard of Jupyter notebook, sentiment analysis, or most of the suggested modules before beginning this Twitter sentiment analysis challenge. My limited Python experience was entirely within physics simulations. I began by reading all the materials on the competition site, then downloading Anaconda so I could begin to experiment with all the new modules. I read dozens of tutorials and example sentiment analyses on related problems, and watched many Youtube tutorials and lectures. Finally, I drew from these resources to begin writing my own code.

**Solution Approach**

The solution began with the basic steps of importing data and relevant modules. The data was examined to understand its formatting, then split into training and validation data. When first developing the models, the default data split of 75% training and 25% testing was used so the accuracy of the models could easily be compared. However, by the end 99% of the data was used for training because the predictions became more accurate with more data and the validation results were less important than the models’ performance for the competition.

The next step was to extract features from the data using various techniques. Bag of Words was used first, as it was the easiest to understand and implement. The competition’s example code greatly limited the number and type of features its Bag of Word program accepted, but it was found that the results improve with fewer restrictions. In trying to limit the noise, the example program lost more of the signal. The next step was cleaning and vectorizing the data and transforming it into document term matrices (or “Bags of Words”) which the different models would be able to understand. Then similar feature data was prepared using a TD-IDF method.

Next it was time to build the models. The first was a Naïve Bayes multinomial model, as it is well suited to natural language processing and runs very quickly. The model was trained on the Bag of Words data, then used validation data to see the accuracy and confusion matrix. The accuracy is not very useful in a heavily skewed data set such as this where the null solution is quite high, but the confusion matrix can be useful to compare different models and later attempt to improve them. Finally, the trained model was used to predict values for the solution file and the results were written to a .csv in the proper format.

The process was repeated for the TD-IDF method, and then again with a Logistic-Regression model. Logistic Regression was selected due to its common use in natural language processing, decent speed, and probabilistic nature. The logistic regression model was different in that it returned predicted probabilities rather than integer values, so if a tweet was 30% or more likely to be offensive it was considered offensive. This value could be tweaked depending on the relative number of false positives.

Finally, a SVM model was built. SVM was chosen last because it is slow, but generally the most reliable off-the-shelf model. It took by far the longest time to train and run but gave the most promising results for accuracy and the confusion matrices.

**Comparing Models**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Naïve Bayes** | **Logistic Regression** | **SVM** |
| Bag of Words | 0.6167 | 0.7318 | 0.7462 |
| TD-IDF | 0.2421 | 0.6182 | 0.7020 |

Table 1: Competition f1 Score for Each Model

As seen in Table 1, the best result was achieved by using SVM with Bag of Words features. It achieved a competition f1 score of 0.7462. Bag of Words outperformed TD-IDF in every model, especially Naïve Bayes by a huge margin. SVM similarly outperformed all the other models with both features. Logistic Regression was the second best by a small margin, and Naïve Bayes was significantly worse than both.

The run time for each model was estimated in the code and is the opposite of the accuracy results. The difference came in orders of magnitude. NB models took only milliseconds to run, whereas logistic regression took a couple seconds and SVM a few minutes. For larger data sets, this could become a major issue and the number of features would likely have to be reduced. However, for such small data sets it was not a problem.

The confusion matrices revealed more differences. Both NB and SVM models had few false positives, but many false negatives. The logistic regression had the most balanced results between false negatives and positives.

**Results**

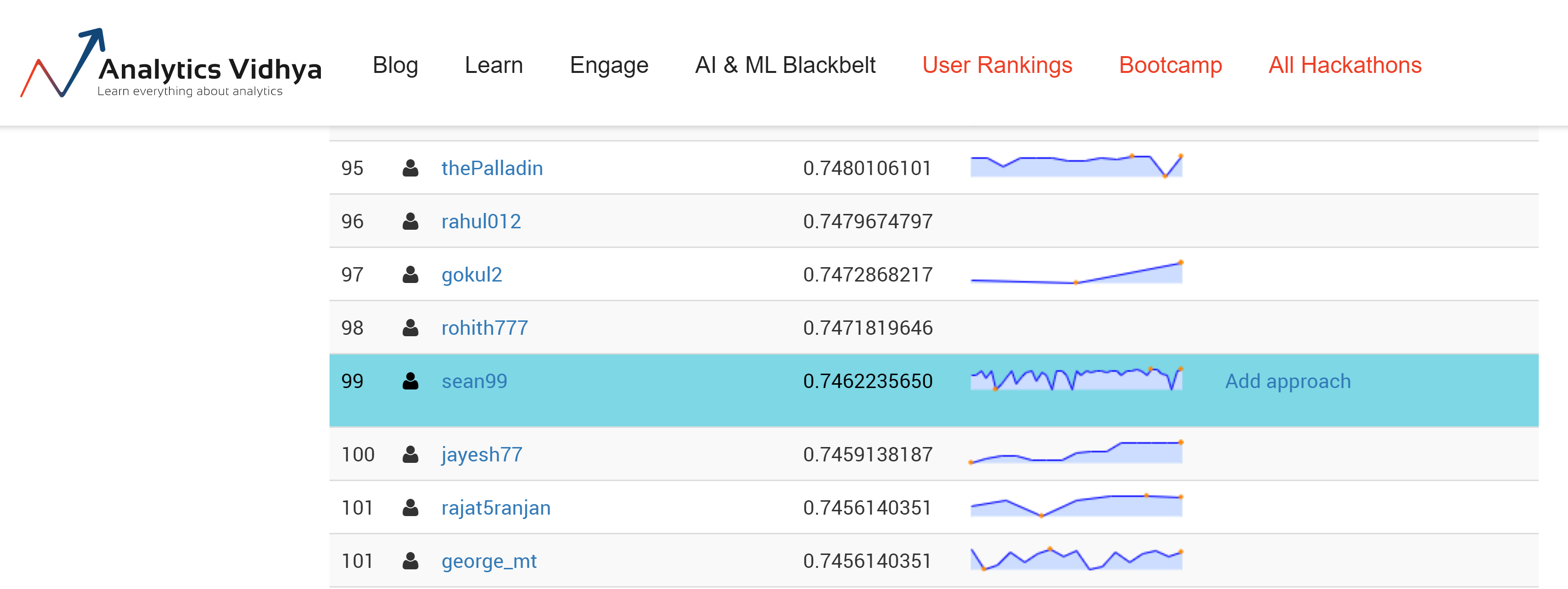


Figure 1: Leaderboard Results

The result of 0.746 was good enough for 99th out of approximately 800 participants in the leaderboard as of Sunday, December 23, 2019.

**Future Steps**

There are many different methods for acquiring features and many different models, all with parameters to be tweaked and optimized. The next logical step would be to put more of each in the program to identify the most promising combinations. Word2Vec has a more elegant evaluation of sentiment to acquire features than either Bag of Words or TD-IDF, and would be the next choice for implementation. After that, Doc2Vec would be the most likely feature method to add. There are plenty more models such as Random\_Forest and XGBoost which can be implemented as well.

After getting to about five feature methods and five models, the most promising three or four should be selected to try for improvement. The best place to start is the major parameters, such as Logistic Regression’s solver, which dramatically impact performance. Then the more minor changes such as feature number and word grams can be tested. Resources such as confusion matrices and checking the false results for patterns can help understand why failures occur and which parameters might help. It is important to use the same random number seed across all trials so that chance has no impact on the results. Finally, when each parameter has been optimized, the final model can be selected.