Jacob Warfield & Chris Johanson

Kaggle Competition Write-Up

Team Name: House of CARTs
Score on Public Leaderboard: 0.833
Code citations: Scikit-learn: https://scikit-learn.org/stable/, Natural Language Toolkit: https://www.nltk.org/

Introduction:

By culminating the class with this project, we got to utilize some fantastic public libraries of machine learning code. In particular, we realized how many good-quality sources exist online and learned a lot about utilizing public code in a private project. In some ways, previous projects have felt like working in a vacuum. It was great for developing our skills and limiting our focus for each model we analyzed, but allowing the use of public code modules opened our eyes to so many more possibilities.

Running Our Project:

The zip file we submitted to CMS contains the sample, train, and test .csv files we were given, this write-up, and then two python scrips. Our classifier generator is called "generateClassifier.py." This looks at the training data and generates a random forest classifier and then looks at the testing data to write our predictions in the specified format to a new "outputTest.csv" file. Additionally, we added a way to check the validity of our classifier without submitting to Kaggle. If generateClassifier.py is run with the keyword "validate" after it, then it instead looks at a portion of the training data called "doubletrain.csv" to generate the classifier and "validation.csv" to create predictions. Then if you run the other python script, "checkValid.py," the terminal will print out the percentage of correct predictions on the 200-sample validation set.

Initial Data Analysis

After looking through the training and testing data, we found that most of the fields were nearly useless. Longitude and latitude had about four valued entries in the 1190 samples. So did the fields like "Truncated" or "Favorited." We were really hoping to find some patterns some of the non-text data, but we did not find much of substance, not even in the "retweet" count. Apparently his staffers produce just as "retweet-able" posts as he does. This actually was interesting, because the posts from his staffers are far less sensational and much-less discussed in the media. But, after modeling charts in excel, no connection seemed relevant. So, we took the training data and, using the csv library in python, read each line of the .csv file into either a List of tweet text or a list of labels. We then passed out our yTr for later use in our model creation and passed the xTr onto our other pre-processing steps.

Pre-Processing Techniques:

So, in our pre-processing we decided to only focus on the samples' tweet text. We felt that only analyzing text was a good way to focus our time and thinking. So, the first thing we did with the data was strip away all the other fields and create an array of the samples twitter text. From there we used an external library NLTK to tokenize the strings. It was really nice because

the library has a method specifically developed for scraping tweet's and handling those special characters. We additionally developed simple regular expressions to quantify the presence of the "#" and "@" characters regardless of text afterword as well as website-links regardless of what the actual URL is. With our data processed we fed it into a method within the scikit-learn external library. This method counted presence of specific words that we were looking for based on our custom vocabulary. The purpose of this was two-fold. One, it made sure that testing and training data both led to the same size matrices of word-count. Two, based on our data analysis and the David Robinson's original blog post, we had a sense of which words were likely to differentiate tweet origins. This scikit-learn method, TfidVectorizer, was really helpful in that it did the word counting and normalization all in one step. This then output a matrix, xTr, we used to train our model.

Features:

Our xTr matrix was a normalized count of the presence of words we defined in a custom vocabulary. The full list has about 40 words/characters in it, but in general we tried to select words that likely came from trump such as ['badly', 'crazy', 'weak', 'funny', 'joke' …]. These words came from our own analysis of the data as well as David Robinson's blog post where we looked at frequency of word use from the two tweet origins. Additionally, we put in "@" because almost exclusively President Trump directly mentions other twitter accounts in his tweets. Meanwhile, "#" and "https" were used to catch staffer tweets. Trump almost never tweets with hashtags and very rarely links to websites or images.

Selection of Random Forests and Hyper-Parameters:

We used scikit-learn's Random Forests model in order to create a classifier. We used random forests because we had just used it in our last project and it was interesting to apply again. The biggest advantage it gave us was it allowed us to easily look at our feature importance as we fine-tuned our vocabulary in order to improve our model. Scikit-learn offers a built-in method to print out feature-importance. During our vocabulary we development we consistently used that method. We left most of the hyper-parameters of the Random Forests method of scikit-learn as default. We adjusted the n_estimators field to change how many decision trees we used for the classifier. After adjusting it to values between 2 – 20, we found no consistent increase in improvement after 10 samples, so we used that. This could certainly be an area of improvement in our model, had we fully delved into all of the hyper-parameters scikit-learn gives us.

Conclusion:

This was a great project to expose us to public libraries for machine learning and show what awesome applications a python script can have in real-world analysis. We both now want to comb through more of the archives of scikit learn to better understand its capabilities as we will likely use it again in the future. We think we could improve our project by taking in some more relevant features such as time of day or catching the few cases weird data like "latitude" and "longitude" have actual data, but we like to think we did well with our public score of around 0.83. This project has also been a huge confidence boost in regards to writing a machine-learning python script from scratch, as we have been working in the Vocareum module for all other projects. I, at least, am looking forward to bragging about this project whenever one of Trump's tweets comes up in discussion in the future.