James Gallmeister

INFO 4604 – Applied Machine Learning

December 18, 2017

Final Project Report

When first starting this project, I had the intention of analyzing tweets from Russian Twitter Trolls to see if there was a way for a classifier to easily identify Russian Twitter Troll accounts, but due to lack of data I had to abandon that idea early on and take up something else. I was still focused on doing something regarding misdirection on social media, so I decided to take on stance detection of articles based on their headlines and article bodies.

The data that I used came from the official fake news challenge as it seemed to be the most complete dataset, unlike some on kaggle.com that may have required me to go through checking for holes in data, etc. On the fake news challenge website, it does not describe how the dataset was procured, but the data itself consists of news articles headlines, bodies and a label. Before I could use the data I had to join the two csv files, one of which contained the headline, bodyID and label, while the other contained the bodyID and the body text. While joining those two and writing it to a third csv, ‘train.csv’ I gave each instance a label as to whether it will be used for training or testing. I trained and tested my classifier on this single csv of around 50,000 instances even though there was a lot more data including unlabeled and specific data, because I wanted to keep the data simple for the sake of time and getting everything done.

As for the features, I used different n-grams on a bag of words that included the headline and body. I utilized the countVectorizer class in sklearn after attempting, unsuccessfully to use the code we used in homework 4b to create our own features. I also utilized transformed the feature vector given by the countVectorizer using the TfidfTransformer so as to implement tfidf into my data, which did increase accuracies overall.

I used three different algorithms, including Naïve Bayes, Stochastic Gradient Descent and Support Vector Machines. I chose these because they are relatively simple to implement, and we used text as features for two of these in different homeworks, and as for the SVM, I read online that they are usually pretty accurate in terms of classifying text. Naïve Bayes was the first algorithm I chose because over the last few lectures, we went over it pretty extensively and it was mentioned to be very good at classifying using text as features, so that was the first algorithm that popped into my mind when beginning this project.

Overall, the algorithms I chose did not do too poorly, but did not do too well either. Using Naïve Bayes, I consistently get around 81% training accuracy and 79% testing accuracy. When using the SGD classifier, I get around 90% training accuracy and 85% testing accuracy. Now, the SVM was a real thorn in my side. When I run it with all instances, it takes around 3 hours to fit the classifier and then more time to score, so I have the user input the number of features to use, and I found that the most reasonable amount to use so as to get OK scores and still be able to use your computer for the rest of the day was anywhere between 1000 and 10000, with 1000 going quickly and 10000 taking a few minutes. Using 10000 features, I get accuracies around 82% for training and 78.5% for testing. I do have an implementation of a gridsearch in each of my function and it is up to the user to decide whether or not he or she wants to utilize them. For the Naïve Bayes and SGD implementations, it takes a good amount of time, but for the SVM it takes significantly more. When I ran the gridsearch of the SVM classifier, I had my parameters array with a *‘gamma’* value, a *‘C’* value and different ‘*kernel’* values, and I left my computer running for 72 hours without the gridsearch object completing being fitted with the training instances. All of my classifiers give the user the option to do a gridsearch or not and they also can look at the confusion matrix as well.

When looking at the confusion matrix, each algorithm does a good job classifying the ‘*unrelated’* stance, but it often gets the ‘*agrees’* stance mixed up with it quite often. As I stated above, each different classifier gives the user the option to see the confusion matrix for the classifier they chose.

As a side note, in my code, the user has the option to run a gridsearch and if they choose not to then the classifier being used already has the best parameters that I could find through testing implemented, but the user does not get any validation accuracy or get to see the best parameters either. Overall, I found this final to be my favorite CS project I have ever done and I am actually going to continue to implement new algorithms and features and try to improve the accuracies. In the github repository, there is unlabeled data, so something that I may be able to implement later on would be some sort of semi-supervised learning using Naïve Bayes as that is the algorithm we learned about that implements semi-supervised learning. This project definitely made me a better programmer and problem solver and was a lot of fun to do!