Abhinav Pottabathula, Drew Kaul, Kanak Garg, Karim Abedrabbo

Data Science Decal

Project 2: Sentiment Analysis Write-Up

*Introduction*

Our original problem was to perform sentiment analysis on a set of movie reviews in order to determine whether overall reviews were positive or negative. We used a bag of words model to featurize the text, using the counts of words as our features. We used logistic regression, decision tree, and a random forest in order to classify our reviews.

*Data*

We started by reading our data into a Pandas DataFrame, in which each row represented a text file and each column contained the counts of each word in that specific text file. We added a column to store the label ‘pos’ or ‘neg’ of the review and a column which stored the file name. All of the NaN values were replaced with 0s within the DataFrame since the count of that particular word was 0. We then randomly shuffled our dataset and then split it into a training set and a validation set, with 80% going to the training set and the remaining 20% going to the validation set. The training set and validation sets were split into an x frame, which contained the words, and a y frame, which contained the classifying label.

*Logistic Regression*

Using sklearn’s linear\_model.LogisticRegression(), we were able to create a logistic regression model. We fitted the model to our training data, and then scored it on that training data. We got a training score of 1.0, which meant that the model overfit on our training data since it was also fitting to the potential noise in our data. We got a validation score of 0.846, which meant that even though the model was overfitting it also scored pretty accurately on the validation set. We changed the parameters by setting C, the inverse of the regularization strength, to 0.008. By regularizing we helped reduce overfitting so that our training score became 0.987 and the validation score was 0.840. We then performed backward stepsize selection to reduce our number of features and thus overfitting. We first calculated the mean and standard deviation of the coefficients, or essentially weights, of our model. We selected the features to remove by removing the columns from our DataFrames which had weights that were over one-third of the standard deviation away from the mean weight. We retrained on the training set and got a training score of 0.957 and a validation score of 0.830. When compared to our original accuracies, we scored almost the same, the only main difference being that our stepwise selection reduced overfitting. This was due to the fact that we removed particular data that we determined to be far away from the mean, or potentially outliers, and thus the model could not fit to some of the potential noise within our data.

We also tried a couple of other methods to reduce overfitting. Instead of removing the columns from our DataFrames which contained weights deemed to be far away from the mean, we simply their respective weights in the model equal to 0. This, however, did not work as well as intended, as indicated by our training score of 0.503 and validation score of 0.488. We also tried adjusting hyperparameters by creating a logistic regression model in which there was an l1 penalty, a C value of 0.01, and a tolerance of 2, but this model had a training score of 1.0, meaning it was still overfitting, and a validation score of 0.577. Finally, we tried using recursive feature elimination, or RFE. Our intuition was that RFE would reduce overfitting since it would start with the initial set of features and keep considering smaller subset of features recursively until the optimal number of features was reached. That optimal set of features would not include the noise (or redundancy) present in our current set of features, thus preventing our model from fitting to that noise. However, we could not find an optimal step and number of features to select, and ended up getting a validation score of 0.670 and a training score of 1.0, meaning we still overfit.

*Decision Trees*

When it comes to figuring out the best parameters for a decision tree for this data, there are just generally a lot of possible of possibilities. What we ended up implementing was using grid search to iterate through lots of combinations of parameters, of which the grid search reported the best three validation scores and the parameters used. One of our goals was to reduce overfitting, and the two parameters that had an influence on reducing overfitting in the training set were max\_depth and and min\_samples. Max\_depth essentially limits how deep the decision tree can be and thus limit the complexity of the decision boundaries on the training data so as not to overfit. Min\_smaples\_leaf specifically changes the model's sensitivity to noise in the training data where a smaller number of sample leaves makes the model more prone to capturing noise. On another note, the results of grid search for a single decision tree were not very helpful because they produced nearly equivalent validation scores to a decision tree with default parameters; the only aspect we were able to optimize to a certain extent with grid search was overfitting. We were also not able to test variations of each and every parameter because grid search is computationally expensive. Our best validation score was 0.58950617284 with a training score of 0.941 and parameters {'criterion': 'entropy', 'max\_depth': None, 'max\_features': 'auto', 'min\_samples\_leaf': 1, 'min\_samples\_split': 10}.

*Random Forest Decision Trees*

In choosing the parameters for a random forest decision tree, we could build off of the information we learned from implementing parameters for the decision tree and omit the parameters that we believed did not make a significant difference. We chose to run grid search on only three parameters, one specific to random forest, and the other two generalized to the decision trees in random forests. One of my parameters was necessarily the number of decision trees I include in my random forest, where most often a greater number of decision trees will result in a more accurate classifier against the validation set. The other parameters we chose to check were max\_depth because it had a very relevant impact on not overfitting the training data, and max\_features which is data specific but generally increases the options to be considered at every node. However, a lot of overfitting is taken care of by the nature of a random forest.

A random forest essentially takes a random fraction of the data and selects only a random fraction of features and builds many decision trees on these different random subsets. What this ends up doing is creating many trees that are of a shallower depth (low overfit) and averaging the results of these trees. It isn't as prone to overfitting as analyzing all features in one decision tree and creating a really complex decision boundary because of its large depth. Our best validation score was 0.821 with a training score of 1.0 and Parameters: {'max\_depth': 100, 'max\_features': 'auto', 'n\_estimators': 1000}. We do understand that increasing max\_depth increases proneness to overfit, however we consistently got the best validation scores when we overfit with a larger max\_depth.

*Conclusions*

We had the best results when using Logistic Regression and the Random forest which both consistently resulted in a average validation accuracy of around the .8’s with the best parameters. However, both these methods consistently produced higher validation scores when we overfit them to the training data. We could change the parameters so that it would not necessarily overfit in either classifier, but our validation score suffered as a result on average. We also did a lot of tinkering around with overall parameters of the different classifiers to figure out which parameters made a significant difference on the scores and which parameters/data directly influences overfitting and why. In a broader scope, we were able to determine the sentiments of movie reviews on overall positivity or negativity through the use of different classifiers, which is a pretty amazing feat.