Rohan Nagar (ran679)

In-Class Kaggle Midterm Report

I. INTRODUCTION

Over the course of the past week, I have made efforts to produce the best predictive model for the In-Class Kaggle Competition. This includes feature exploration, training many different models using cross validation and searching over parameter sets, and ensembling models together. Each of these tasks are discussed in more detail in the following sections, along with what worked and what did not work in each. My Kaggle details are as follows:

- **Name:** Rohan Nagar
- **Kaggle Name:** RohanNagar
- **Best Public Score:** 0.86561

II. FEATURE EXPLORATION

The first thing that I did when starting the competition was to do some feature exploration so that I could get a better understanding of the data and its features. The first thing I did was a simple `df.describe()` on the Pandas data frame of the CSV file. This was able to show me a description of the features, including a count, the mean and standard deviation of each feature, and the min and max values. This was valuable because I was able to see the range of values for each feature, as well as if there was any missing data (which there was).

Next, I plotted each feature's distribution to get a better sense of how the features were distributed. I also plotted the distribution of the target classification variable. I again gained valuable information from these plots. Upon plotting the target variable, I was able to clearly see that the classification problem would be lopsided. There are many more samples with a class label of 0 than there are with a label of 1. Additionally, I could see which features were distributed more evenly or in more Gaussian-like distributions. These features would likely be the most valuable in training models.

I also created pair-plots of each pair of features. I did this in order to determine which features were highly correlated with one another. From this, I was able to see two pairs of features with high correlation - it looked like you could fit a straight line almost perfectly between them. Since these pairs were so correlated, I removed one feature from the data set from each pair. Including both when training the models would be redundant.

Finally, I attempted to use PCA to find the features that best described the data. I used Scikit-Learn's PCA class. However, PCA did not seem to work as expected. After running PCA with the following code, I saw that the explained variance started at 100% from the first principal component. This didn't seem to make any sense, so I stopped attempting to use PCA.

```
pca = PCA()
X_reduced = pca.fit_transform(X_train)

# Show how much variance is explained
np.cumsum(np.round(pca.explained_variance_ratio_,
decimals=4)*100)
```

III. MODEL TRAINING

After feature exploration, I began to train single models using cross validation and parameter tuning. This was done largely using Scikit-Learn's `GridSearchCV`. Each of the models that I attempted are described in detail below.

A. Logistic Regression

The first model I attempted to fit was a Logistic Regression. This is one of the more simple models, and so is the first that I tried to fit. I performed a grid search over different values of the `C` parameter, and tried both L1 and L2 penalties. After running grid search with cross validation, a `C` value of 0.1 with L1 penalty performed the best, giving a score of 0.692 with cross validation. Of course, we can do much better than this, so I began to explore other models.

B. XGB (Extreme Gradient Boosting)

The next model I tried was an XGB model. Actually, this is a boosted model so it is more of an ensemble approach, but I am considering it as a model here because I did not have to do the ensembling myself. I performed a grid search over different values of max_depth, learning_rate, n_estimators, min_child_weight, and colsample_bytree. This took quite a while to run, but after the search finished my best score was 0.860. This is performed much better than the logistic regression.

C. Random Forest

Next, I went with a Random Forest. This again is more of an ensemble model with many decision trees, but it is one package in Scikit-Learn. Running grid search was taking a very long time, and I wasn't sure if it was ever going to finish. Instead, I tried adjusting the parameters myself and running a single model with cross validation. I ended up using the parameter values of n_estimators=2000, criterion='entropy', max_features='auto', and bootstrap=True. These gave me a public score on Kaggle of 0.86045.

D. AdaBoost

The AdaBoost model fits boosted classifiers where the weights of incorrectly classified samples are adjusted so that future estimators can favor those more. I trained this and it worked very well for me also. Performing a grid search on the n_estimators and the learning_rate, I was able to achieve a cross validation score of 0.855. This further improved on the Kaggle public leaderboard to a score of 0.86095.

E. K-Nearest Neighbors

The K-Nearest Neighbors model is the next one I tried, and it did not work as well as I had hoped. After searching over the parameters, the best cross-validation score that I got was 0.615. This is even worse than the Logistic Regression. Perhaps this is because the neighbors of each data sample don't accurately describe how the point should be classified. Another problem could have been that I was using Euclidean distance as the distance metric. KNN likely would have performed better with a better distance metric, but determining an appropriate metric is a difficult task.

F. Naive Bayes

Next, I tried a Naive Bayes model. In this model, you assume independence between all of the features. In this data set, that is a bad assumption, but I wanted to try to run the model anyway to see how it would perform. As expected, the cross-validation score was 0.5176, which is very bad. I did not use this model in any ensembling method.

G. TensorFlow Neural Network

Finally, I attempted to use a neural network from TensorFlow. Using the package skflow, I used a TensorFlowDNNClassifier to attempt to classify the data. I tried this with both a single hidden layer of 5 nodes and with 3 hidden layers of 5, 3, and 4 nodes. Unfortunately, neither of these attempts were successful, with a cross-validation score of around 0.501. This was even worse than Naive Bayes.

IV. ENSEMBLING

Finally, the last step in the machine learning pipeline is to ensemble our models together. In this step I tried various techniques including ranked average and stacking.

A. Ranked Average

After reading about various ways to ensemble models, I came across one that uses the outputs of the models. This is advantageous because the models do not have to be re-trained for the ensemble. I was able to implement it with help from this guide: http://mlwave.com/kaggle-ensembling-guide/. I performed a rank-average with the models I had trained previously, except for K-Nearest-Neighbors and Naive Bayes. However, I was surprised to see that the public score I achieved on Kaggle was lower than my scores with AdaBoost or Random Forest. The public score on Kaggle was a 0.84922.

B. Stacking

In an attempt to better ensemble the best models that I trained in the second step, I tried stacking. This is where I took the results from a previous model, added it as a feature to the training data, and then fed that to train the next model. This worked very well and I was able to move up on the public leaderboard quite a bit. My final stack was a Logistic Regression, an AdaBoost, a Random Forest, and lastly an XGB. This stack produced a score of 0.86561 on the public leaderboard, which was my best score. I also tried stacking in a different order, but that did not produce a high score. It actually was worse, so I believe that I probably made a mistake while training the models. I've learned that stacking works very well in improving the overall score.

V. CONCLUSION

In conclusion, there were three main steps to performing in this Kaggle competition: feature exploration, single model tuning, and ensembling. By taking time at each step and trying out multiple models (some that didn't work and some that did), I was able to improve my score on the leaderboard by quite a bit. I learned more about how each model performs and where each model's strengths are. I also learned that stacking can greatly improve your overall score on the testing data.