Santander Customer Transaction Prediction Assignment

Spring 2019

Brenten Canfield

As a final result with my machine learning set up, my current roc_auc_score on Kaggle is 7.773. And my results, when running a cross evaluation on a subset of the provided training data is as follows:

```
ROC AUC Score: 0.8512720884275355
Accuracy scores: [0.76893146 0.76923077 0.78006187 0.76908492]
Confusion matrix: [[15199  4791]
                   [4356 15742]]
Recall Score: 0.7718269806425863
Precision Score: 0.7718269806425863
```

These cross evaluation results are from a split of 5 different subsets, of the subset, of the provided training data. Clearly the cross evaluation results were much better than the final score on Kaggle's test data though.

To explain why I am only using a subset of the provided training data, I first must explain how I have processed the data. I have taken the entire training set, and split it into nine different subgroups. In each subgroup, the positive (customer made a transaction) rows are duplicated, and the remaining negative (customer didn't make a transaction) rows are uniquely, and randomly selected. This results in the aforementioned subgroups having an equal amount of rows, again with all unique negative rows, and all of the duplicated positive rows. This kind of grouping seemed necessary since the data set suffers from the Imbalanced Classes Problem.

In order to consider the influence of all the training data, I trained nine separate models, each on respective subgroups for the training data. Each model was then saved to a .joblib file. When it came time to perform predictions on the Kaggle provided test data, I would then load each model for the task. Each model would perform predictions on the entire test data. To combine the results, my models performed a soft vote on whether the prediction was best as a positive or negative result. This required,

for each predicted row, averaging each model's probability scores (probability of whether the row was positive or negative). I tried having ties, where a soft vote resulted in equal probabilities, default to negative, and then the converse. My score on Kaggle's provided test data seemed to favor defaulting to negative predictions on probability ties.

I ended up choosing Scikit Learn's RandomForestClassifier ensemble as my model. I also tried AdaBoostClassifier ensemble with the estimator as a DecisionTreeClassifier. But the results that I was seeing from the RandomForestClassifer ensemble seemed much more promising. Both potential model choices were evaluated using a pipeline, where the pipeline consisted of a data scalar, namely StandardScalar, then the respective model.

Once I realized that the RandomForestClassifier seemed more promising, I ran a cross evaluation grid search, using GridSearchCV, to find the best hyper parameters. The following is an example of the parameters search on:

```
{ "Model__criterion": ["gini", "entropy"],
"Model__n_estimators": [10, 50, 100, 300],
"Model__max_features": ["sqrt", None],
"Model__bootstrap": [True, False] }
```

It seemed to be best to set the criterion as "entropy", and n_estimators as high as computation time would allow. My final result was from setting n_esimators at 700. The max_features, and bootstrap hyperparameters seemed to not be relevant enough to worry about. I'm sure there are other hyperparameters worth considering, but I don't think I understand them enough to know what to experiment with. It was also difficult to experiment with hyperparameters, give the large size of the training data, which made computation just take too long.

Again, due to the large size of the data provided, it was difficult to run and try many different models and hyperparameters. I suppose that is fairly representative of real life problems in machine learning though. I would like to learn more about how to deal with such large datasets. As a side note, if I have time, I would like to try to just duplicate the positives in the training data, rather than training separate models on the

aforementioned subgroups. This may or may not help with the final result on the Kaggle test data, and will certainly be harder to do cross evaluation on though.