ICS635 Homework 3

Lambert Leong

March 14, 2019

1 Introduction

I chose to participate in the Santander Customer Transaction competition. A quick survey of the raw training data revealed that there are 200 features which contain positive or negative float values. My initial thought was to perform a principal components analysis (PCA) to try and reduce the dimensionality of the data. Several visualization kernels showed that the distribution for each class with respect to each feature, i.e. 'var_0', 'var_1',etc..., Gaussian and is not very different from each other. In fact, two classes overlap significantly for almost all features and determining any separability appears to be non-trivial. This led me to believe that mapping the data into a new space with PCA may not be useful for the purposes of reducing data dimensionality.

I first looked at ensembling methods and used gradient boosting to classify the data as is, with all 200 features. I then used the same gradient boosting classifier, with the exact same hyper parameters, on training data which had the dimensionality reduced via PCA. Comparing the results of the original data, with 200 features, to the PCA transformed data, with less than 200 features, indicated that I should not proceeds with PCA and models generated with all 200 features were able to generalize to the validation set better.

The instructions on the competition website stated,"...identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted." From these instruction I hypothesized that each feature could represent a transaciton from an individual's transaction history and I looked into treating the data as sequence data. I explored two neural network architectures which include a long short term memory (LSTM) recurrent neural network and a 1D convolutional neural network (CNN). The motivation behind implementing these neural networks was to try to explore patterns in the sequence of transacitons that may correlate to a particular class.

The instruction also noted that the transaction amount is not really relavent. Each of the 200 feature fields contained either a positve or negative number which could indicate money coming in and money going out, respectively. Under this assumption, I sought to capture the amount of times money came in and the number of times money went out for a particular individual to see if it had any correlation to a particular class. In other words, I captures the total amount of postive values and the total amount of negative values for each individual in the dataset.

2 Methods

First, the training data was split into a training and validation set. The Scikit-learn train_test_split function was used to split the training data. A random_state of 18 was used and the test size was

0.2 which resulted in 80% of the original training data to be used for training and the remaining 20% to be used for validation.

2.1 PCA

PCA was performed using Scikit-lean's decomposition module. PCA was performed on centered and non-centered data which yielded similar results. I sought to reduce the number of features by keeping those which explain 95% of the variance. It turns out that 118 features explain about 95% of the total variance which is more features than what I initially expected.

2.2 Gradient Boosting

The sklearn xgboost module was used to create the gradient boosting model. The model was trained and evaluated on the original training data, with all 200 features, as well as with the PCA reduced data with 118 features. Area under receiver operation characteristic (AUROC) was used to evaluate how well the model was at generalizing to the validation test set. The non-reduced data, the original data yielded higher AUROC values, at around 0.885, when compared to the models trained on PCA transformed data. Different numbers of features were experimented with for PCA.

2.3 Feature Engineering

Functions were written to iterate through each row or record and count the number of positive values and the number of negative values. These two numbers were appended to the end of the dataset as two extra features. Another function was written to capture the longest sequence, within each row, of positive numbers and negative numbers. These features were engineered under the hypothesis that this could be sequence data as well as the assumption that the order of the features have not been randomized.

2.4 Parameter Tuning

Model parameters, which include n_estimators, max_depth, and the learning rate were chosen with the help of the GridSearchCV. GridSearchCV exhaustively searches through a range of chosen values for each of the model parameters and helps you choose the best parameter values. Parameters and their final values are as follows: n_estimators = 1000, max_depth = 3, and the learning rate = 0.3. The tree_method was gpu_hist and predictor was gpu_predictor; these parameters were not tunned and chosen so that GPU acceleration could be enabled.

3 Results

Various visualization kernels on kaggle showed great similarities between the two classes with respect to a particular feature. No one feature seemed to show any significant amount of separability between the two classes however, it appeared that some features were slightly more separable than others. It was thought that PCA would pick out the features that were slightly more separable and lead to better classification. Figure 1 shows the relationship between the number of principal components and the variance explained by those number of components. I had hoped that the

majority of the variance, about 95%, was explained by a few components however this was not the case.

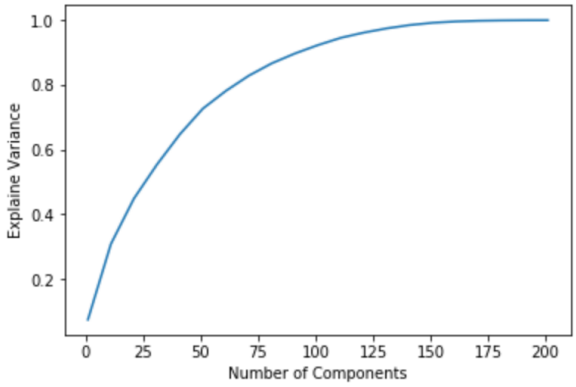


Figure 1: Number of principal components and the explained variance

I experimentally determined if it was worth proceeding with PCA. Using the XGBoost gradient boosting module I defined a set of consistent hyper parameters, which are shown in Table 2for comparison. I trained the model using the training data, 80% of the full training dataset, and varied the amount of components used during training. Predictions were made on the validation set, 20% of the original training dataset, and the AUROC were calculated. Figure 2 shows how varying the number of principal components affects the AUROC score.

Table 1: XGBoost parameters

Parameter	Value
n_estimators	1000
tree_method	gpu_hist
predictor	gpu_predictor
max_depth	3
learning rate	0.3
eval_metric	auc

The results in Figure 2 led me away from PCA. Using all the components with XGBoost resulted in a fairly good AUROC at 0.887. I decided to stick with the XGBoost classifier and proceed with feature engineering. As mentioned in the previous section, I constructed two extra features. Frequency distributions for each of the features are plotted in Figure 4 and are colored by class.

Figure 4 indicates that there are some differences between class 0 and class 1 when looking at the total number of positive feature values and the total number of negative feature values. The separation seen in Figures 4a and 4b display more difference and less overlap than any of the 200 original features. As a result, I hypothesized that adding these features to the model may help with training.

Distributions of the longest positive and negative sequences with respect to each class are shown in Figure 4. These two features were constructed under the assumption that the features correspond

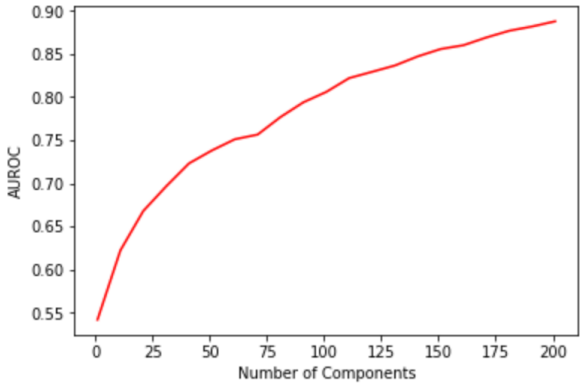
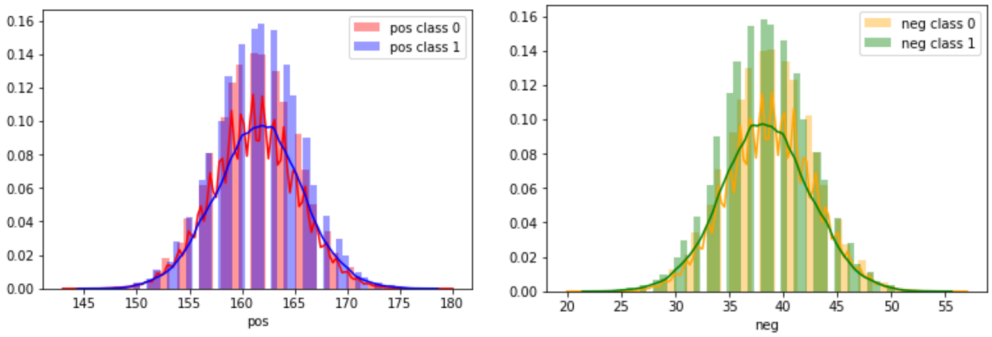


Figure 2: AUROC scores have a positive correlation to the number of principal components



(a) Distribution of positive numbers by class

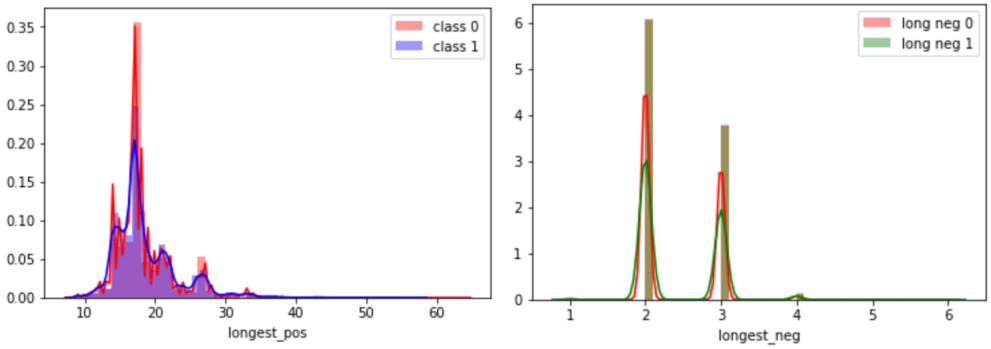
(b) Distribution of negative numbers by class

Figure 3: Frequency distributions for positive and negative values colored by class

to some type of sequence data. Again, I was trying to find features which show more separation between the two classes than the original 200 features.

The resulting distributions for the longest positive and negative sequences are different than the total count of positive and negative numbers as well as the rest of the 200 features There are slight differences with respect to each class which may help to improve the models. Using this new feature may come with some risk because it is likely that the feature order may have been randomized before the datasets were released. If that was the case, the longest positive and negative sequences will be different and possibly irrelevant.

The hyper parameters for the XGBoost classifier were tuned using GridSearchCV from sklearn. GridSearchCV does not support GPU tree_methods and so the search was done on the CPU tree methods. Final models were constructed using the GPU versions of the hyper parameters for speed gains. Finalize parameters are shown in Table 2. Some extra tuning of hyper parameters were done to account for possible differences in the tree_methods between the CPU and GPU implementation. The best hyper parameters were chosen based on AUROC scores for the validation set. It is likely that overfitting could have occurred and that good performance on the validation set does not indicate a good ability for the model to generalize to the test set. In order to mitigate some



(a) Distribution of positive numbers by class

(b) Distribution of negative numbers by class

Figure 4: Frequency distributions for longest positive sequence and longest negative sequence values colored by class

overfitting issues I used cross validation. Overall, the AUROC scores from validation were similar, out to four decimal places, to the resulting public score. This made me confident that any overfitting that may have occurred was not too extreme.

Table 2: XGBoost final hyper parameters

Parameter	Value
n_estimators	3500
tree_method	gpu_exact
predictor	gpu_predictor
max_depth	2
eval_metric	auc

When the hyper parameters were finalized, the XGBoost classifier was trained on the full training dataset using the hyper parameters from Table 2. The classifier was then used to make predictions on the test set. The predictions were uploaded and the public score is shown below in Table 3

Table 3: Final public results

Model	Best Public Score
1D CNN	0.833
LSTM	0.829
PCA XGBoost	0.882
XGBoost w/ feat engineering	0.899

In addition to XGBoost, PCA, and some feature engineering, I implemented a 1D CNN and an LSTM as an exploratory experiment. The initial submissions for both the 1D CNN and LSTM resulted in the scores seen in in Table 3. In an attempt to achieve higher scores for both models I ran then through more epochs. While their performance on the validation sets increased considerably with more epochs, I suspected overfitting to be an issue. The performance on the public leader

board dropped and was lower than the previous models which were constructed on less epochs. These models were not pursued any further. Feature engineering along with the XGBoost classifier for which hyper parameters were tuned gave me the best public score.

4 Conclusion

As of 3/14/2019 the top score on the leader board is 0.924. My best score is 0.899 which was the result of the XGBoost classifier with some added features. It is hard for me to have a confident idea of my models ability to win because of the fact that still little is known about the data, the 200 features, and what each class represents.