Credit Card Default Prediction Using XGBoost, Neural Network, and PyCaret

Jiarui Wang

***Abstract*—This paper focuses on making predictions about whether new credit card customers will default using machine learning techniques. We utilize data with 25,128 credit card users obtained from Kaggle. We first use the python package Pycaret to show the overall performance of the different methods. Then, we apply Neural Network and XGboost to achieve better performance by tuning the hyperparameters. Considering the model explanation ability, computation cost, model complexity, XGboost outperforms in all models.**

***Index Terms - Binary Classification, XGboost, Neural Network, PyCaret***

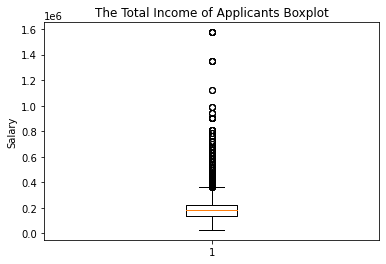
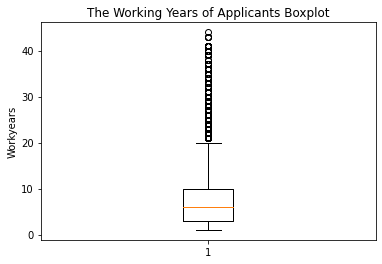
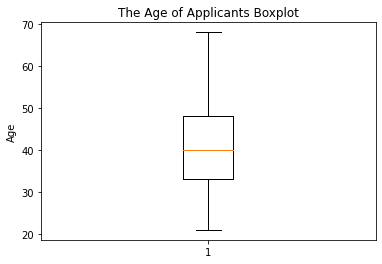
1. **Introduction**

In the financial sector, credit scoring is a widely used risk management technique. The bank can determine whether or not to provide the applicant with a credit card based on the personal information and data provided by credit card applicants to minimize the risk of future defaults. Our task is to utilize machine learning models to classify if an applicant is a 'good' client who will not default or a 'bad' client who will default based on their personal information. We decide to use the PyCaret package to see the overall performance first, and then choose the most interesting and challenging method, namely Neural Network and XGboost to predict the label of the applicants.

1. **Data Description and Preprocessing**

The data we use is “Credit Card Approval Prediction” from Kaggle by Seanny. The Definition of 'good' or 'bad' client is not given directly in this data set. The data set contains a file named “application\_record,” which has credit card users’ background information. Another file named “credit\_record” has past default records of each user. Caesar on GitHub cleaned and transformed both data sets, and merged them into one. He imports the application record file in the ETL software, calculates applicant age and how long applicants have been working (in years), filters applicants that are less than 21 years old, and filters applicants with null/empty values. Based on past default records, he calculates the total number of bad debt and good debt of each customer. If the applicant’s total bad debt is equal to or greater than one, we will label him/her as “default” and marked as 1. Otherwise, we will label him/her as “not default” and marked as 0. The applicant who is labeled as “default” should not be authorized to use a credit card. This label is named “Status1” in our data file.

The columns of the final data are “Applicant\_Gender”,“Owned\_Car”,“Owned\_Realty”,“Total\_Children”,“Total\_Income”,“Income\_Type”,“Education\_Type”,“Family\_Status”, etc. In total, there are 17 features that we are interested in. 7 of them are numerical, 4 are binary, and 6 are categorical. The distribution of some features are presented below. The number of males is 9501, while the number of females is 15627. The average working years of the applicants is 7.69 years, and they own an average of 0.4183 cars. The average income of the applicants is 194836.5. Further processing of the data will be discussed in detail in the following sections respectively for different models.



*Figure 1. The General Description of the data*

1. **Experiments and Results**

In this section, we will discuss in detail the experiments on the task using three different machine learning techniques along with their corresponding results.

**A) Pycaret**

*1. Data Feature Processing and Model selection*

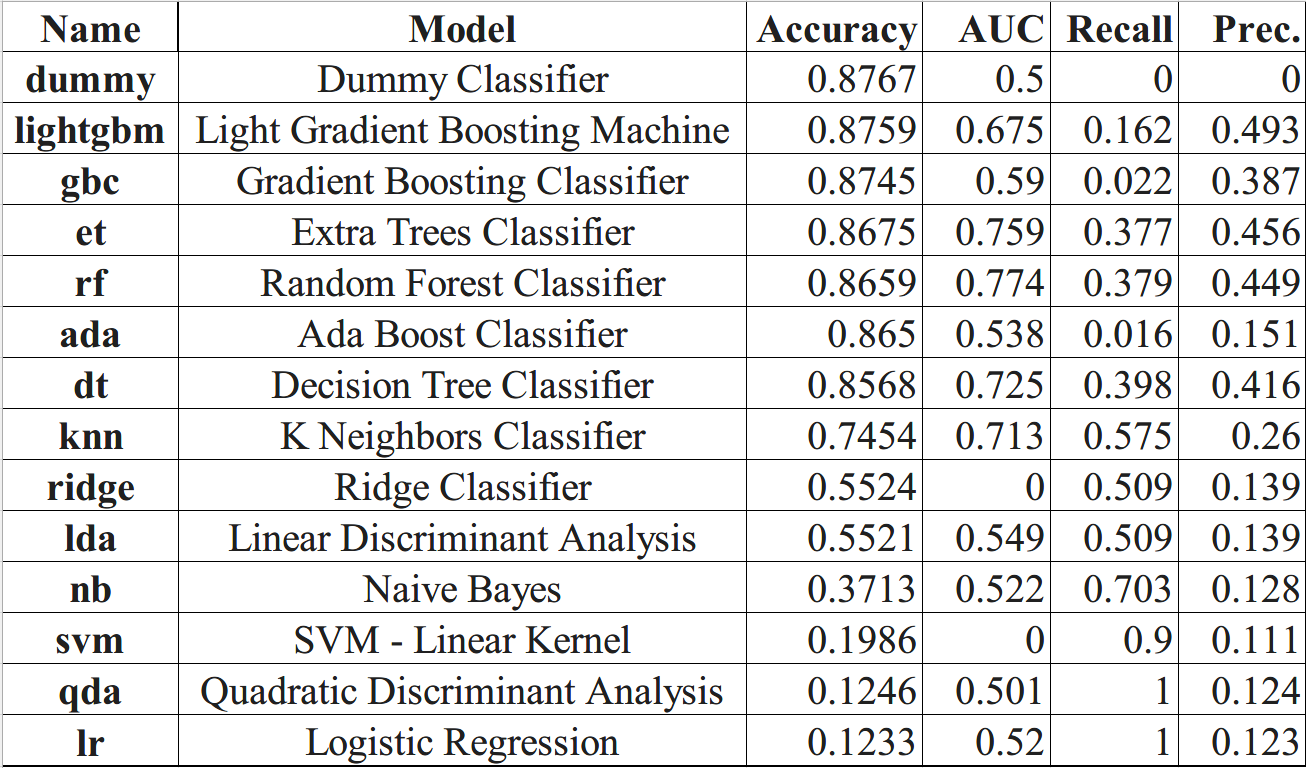
PyCaret shields specific usage details, such as different calling methods of various libraries in modeling, drawing, and feature sorting. It provides end-to-end unified development tools to the outside world. The PyCaret will help us to select most models to train, test, and validate.

The way the PyCaret differs from the traditional method is not necessary to consider the imbalance in the dataset. Setup will consider the problem of imbalanced data. It is totally a Black-box setup. We use only one setup function to prepare the data in this section. This function initializes the training environment and creates the transformation pipeline. The function will take two mandatory parameters: data and target. After setting up, we can make a conclusion by comparing the model, plotting the picture, and tuning the parameter.

*2. Method Summary*

In this section we add the PyCaret to show a summary of the result of the modelPycaret encapsulates tools such as Dummy Classifier, LightGbm, GBC, etc. It includes almost all the usage scenarios and methods of machine learning. Among them, the most abundant support is classification and regression.

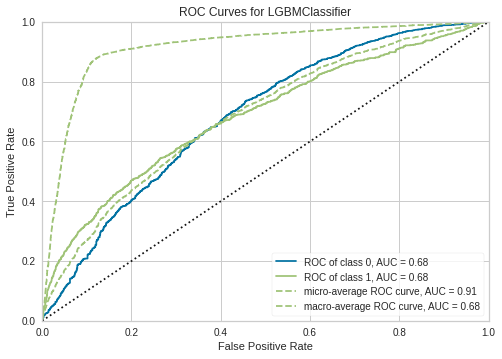
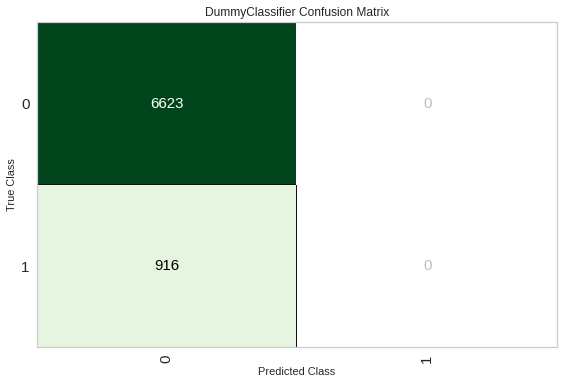
From Figure 3, we know that the top three methods are dummy, lightgbm, and GBC. Although there are some misalignments with the method we applied before, the possible reason is the tuning of the hyperparameter.



*Figure 2. Model Summary*

*3. Result evaluation*

The Dummy variable reaches an accuracy of 0.8768, the confusion matrix shows that true-positive reaches 6623, which is quite a decent result.

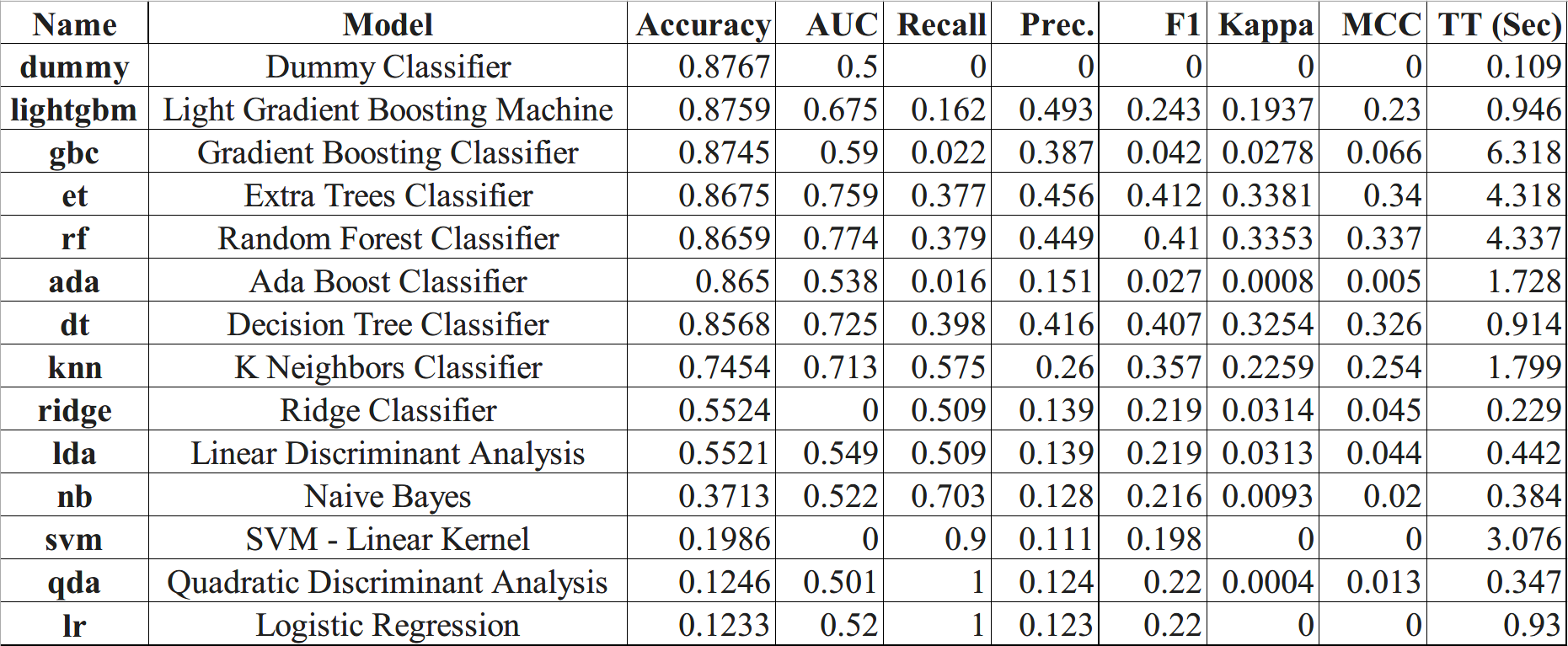


*Figure 3. The DummyClassier Confusion Matrix*

*Figure 4. Roc Curves for LGBMClassifier*

Other than the Dummy Classier method, we decide to focus on LGBM Classier, which is a commonly used method in both industry and academia.

During this process, the tuned LGBM method can reach the accuracy of 0.8759, through the tuning function, the accuracy can reach 0.8771, with a 0.14% percent increase. The standard deviation of the tuning model shows stability. The AUC of ROC of class 1 is0.68, while the AUC of micro-average ROC curve =0.91.



*Figure5. Tuning Process of Lightgbm*

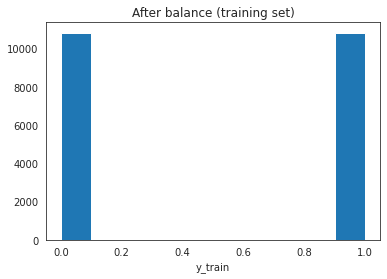
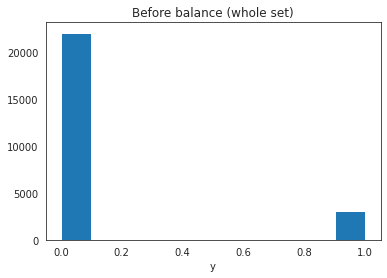
**B) Neural Network**

*1. Data Processing*

We need to further process the dataset we introduced in the above section so that it can be used as the input for neural network models. In total, there are six categorical features in our dataset, which are “Applicant\_Gender”, “Income\_Type”,“Education\_Type”,“Family\_Status”, “Housing\_Type”, and “Job\_Title.” To encode these features, we use the one-hot encoding so that each category of each feature is assigned a dummy variable. If the sample belongs to this category, it will be marked as 1. Otherwise, it will be marked as 0. After the one-hot encoding, the total number of features is 52.

In addition, we need all features to have similar ranges to avoid the model being biased to certain features. Therefore, we conduct normalization on the dataset. We use and compare the two most popular normalization functions, namely StandardScaler and MinMaxScaler. By using the two functions on the same data and the baseline neural network mode that will be introduced in the following parts, we find StandardScaler performs better than MinMaxScaler as it has a higher validation accuracy. To ensure consistency, we will use data normalized by StandardScaler in all of our following neural network models.

Furthermore, it is important to notice that the dataset is heavily imbalanced. In total, there are 25128 samples in our dataset, while only 3085 of them are labeled as 1, which are defined as “default.” From the figure below, we can see that the positive class only accounts for a small portion of the whole dataset. The imbalance of the dataset will largely affect the training of neural networks because the weight of positive samples is low and there will not be enough positive samples to feed the model and facilitate learning. To deal with this issue, we apply the Synthetic Minority Oversampling Technique (SMOTE). This technique is able to oversample the examples in the minority class by synthesizing new examples. It can randomly select “examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line” (Brownlee, 2020). However, as these new examples are synthetic, they should not be used to validate and test how our model can be generalized to original data or real-life data. As a result, we split the dataset into training, validation, and testing sets before applying SMOTE only on the training set. From the figure below we can see that we obtain a balanced training set with around 10000 samples for each class.



*2. Evaluation Metrics*

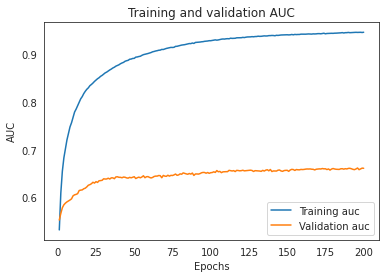
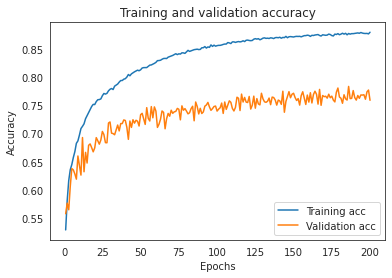
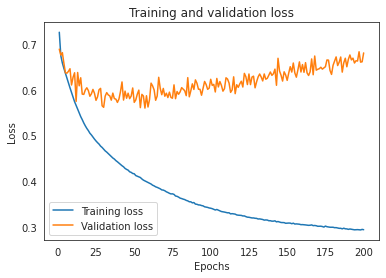
As our task is binary classification, we use Binary Cross Entropy as the loss function. We also include accuracy and AUC in our validation and testing process as metrics to decide how well our models perform.

*3. Models Design and Major Results*

Base configuration

We start from building a simple baseline model with 2 hidden layers. We have 32 neurons in the first hidden layer and 16 neurons in the second hidden layer. The activation functions in the hidden layers are ReLU, which empirically works well. The activation function in the output layer is Sigmoid . This allows the model to output a number between 0 and 1, which is recognized as the probability of default. We use Adam as the optimizer because it also empirically works well in neural networks. We set the batch size to 512 and epochs to 200 as it allows enough iterations for the model to go through the whole dataset and update parameters. Other combinations of batch size and epochs are tried, such as (32, 100) and (1024, 400), but no significant difference is found. Therefore we will stick to (512, 200) in all models. The results at 200 epoch are presented below. It is worth mentioning that, because the distribution of the validation set follows that of the original data while the training set does not, we should not compare the absolute values of the metrics but their trend. We can observe that the validation loss increases while the training loss decreases, which indicates possible over-fitting issues. The validation accuracy and AUC also suggest that the model does not work well.

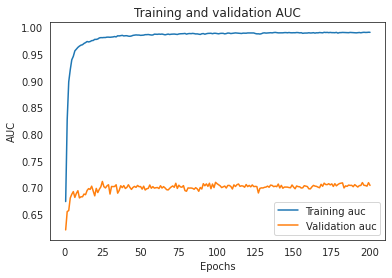
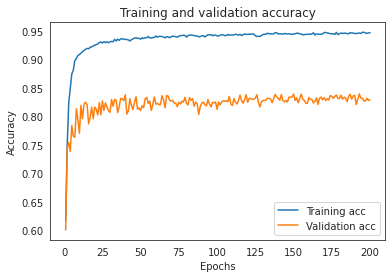
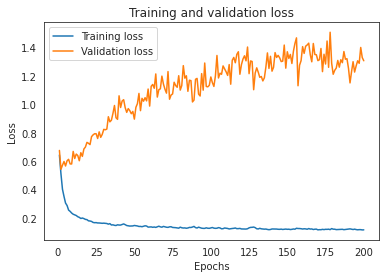
|  |  |  |  |
| --- | --- | --- | --- |
|  | Loss | Accuracy | AUC |
| Training | 0.2940 | 0.8800 | 0.9470 |
| Validation | 0.6808 | 0.7591 | 0.6609 |



b. Deeper NN

We enlarge and deepen our initial model by increasing the number of hidden layers to 5. The neurons of each layer are 512, 512, 256, 256, 128. All other settings are identical. The results at 200 epoch are presented below. As our neural network becomes deeper, validation accuracy and AUC shows better performance. However, the validation loss still increases while the training loss decreases, which suggests possible overfitting. The validation loss also fluctuates a lot.

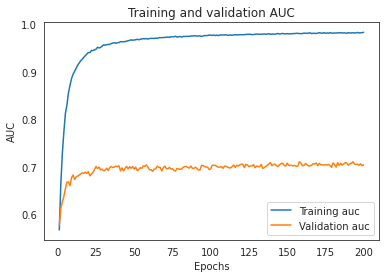
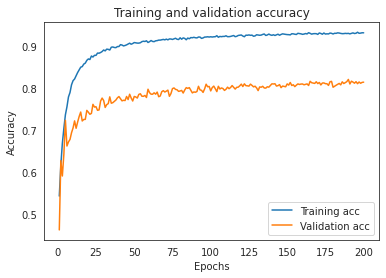
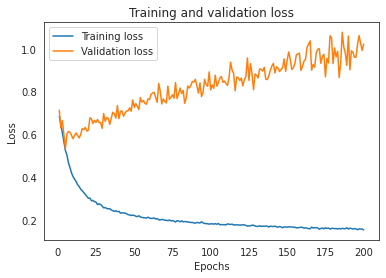
|  |  |  |  |
| --- | --- | --- | --- |
|  | Loss | Accuracy | AUC |
| Training | 0.1155 | 0.9467 | 0.9907 |
| Validation | 1.3073 | 0.8293 | 0.7037 |



c. Regularization with Drop-out

We apply drop-out to deal with overfitting. The model “randomly drops a portion of neurons in a layer in each epoch during training, which forces the remaining neurons to be more versatile” (Majewski, 2020). The validation loss still increases when training loss decreases, but it is lower and less volatile. The validation accuracy and AUC, on the other hand, are stable and do not decrease with the loss. One possible explanation is that, when the model starts to overfit and predict some samples very wrong, it is meanwhile still learning some pattern that can be generalized. Because the binary cross entropy function penalizes bad predictions “much more strongly than good predictions are rewarded,” the effect is amplified when wrong predictions are predicted more and more wrong in terms of probability. However, since the percentage correctness of the prediction does not change, the accuracy and AUC are stable.

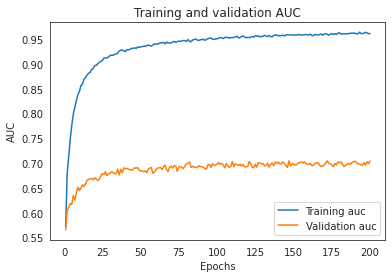
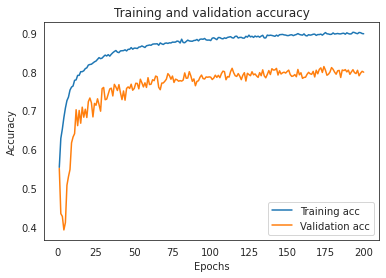
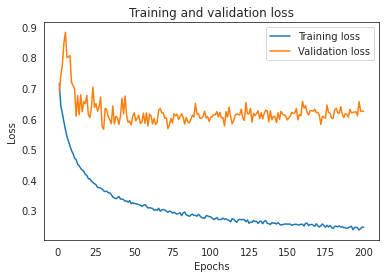
|  |  |  |  |
| --- | --- | --- | --- |
|  | Loss | Accuracy | AUC |
| Training | 0.1550 | 0.9328 | 0.9836 |
| Validation | 1.0209 | 0.8152 | 0.7051 |



d. Batch normalization and Leaky RELU

Batch Normalization and Leaky RELU are good ways to deal with dying/exploding neurons, which in turn help reduce the risk of vanishing/exploding gradients (Majewski, 2020). By changing activation functions to Leaky RELU and adding Batch Normalization for the first 4 hidden layers, we observe a more converged and stable validation loss curve.

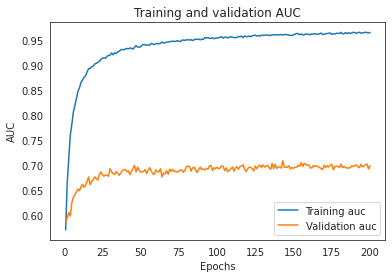
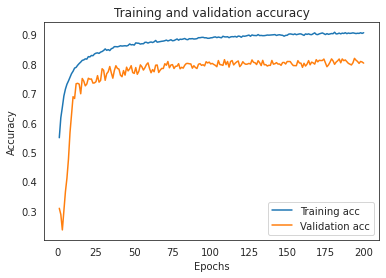
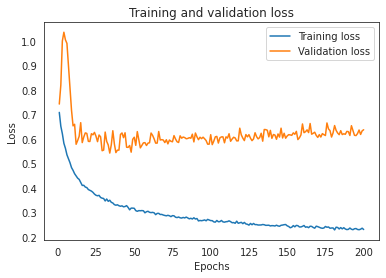
|  |  |  |  |
| --- | --- | --- | --- |
|  | Loss | Accuracy | AUC |
| Training | 0.2457 | 0.8986 | 0.9614 |
| Validation | 0.6236 | 0.7993 | 0.7040 |



e. Larger network

We try to enlarge and deepen the network one more time by adding one more hidden layer with 1024 neurons. No significant changes in metrics and pattern of the curves are found compared to the previous model. We will not utilize this layer.

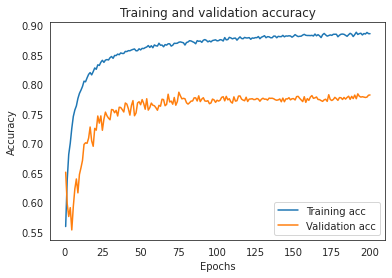
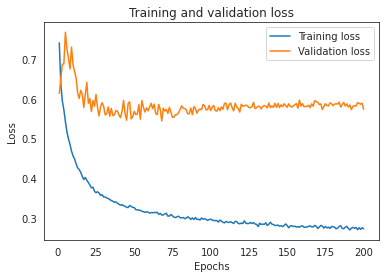
|  |  |  |  |
| --- | --- | --- | --- |
|  | Loss | Accuracy | AUC |
| Training | 0.2303 | 0.9064 | 0.9664 |
| Validation | 0.6373 | 0.8029 | 0.6990 |

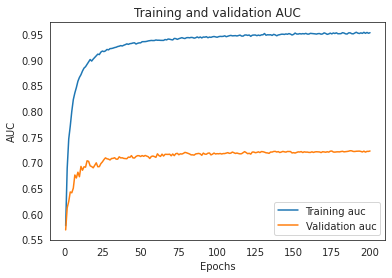


f. Adjusting learning rate decay

We use learning rate decay to train the network faster at the beginning and then decrease the learning rate to make training more precise at the end. We set the starting rate to 0.005, and try 0.001 and 0.0001 as the ending rate respectively. We find using 0.001 returns better performance. Overall no significant improvement is found compared to previous models. This is our final model.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Loss | Accuracy | AUC |
| Training | 0.2590 | 0.8919 | 0.9587 |
| Validation | 0.5998 | 0.7919 | 0.6944 |

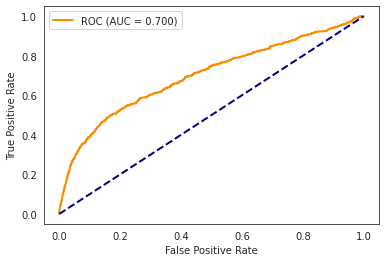
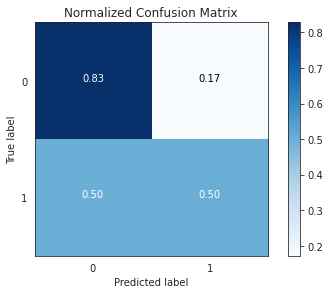




*4. Testing and Analysis*

Applying the final model on the testing set, we obtain the following result. The accuracy and AUC suggest that the neural network does not perform very well on this task given the current data. The precision for “default” is 0.29. Precision for “not default” is 0.92. Recall for “default” is 0.5. Recall for “not default” is 0.83. Notice that “not default” makes up 88% of the original data. If we use this model to decide whether banks should issue credit cards to customers, 92% of the credit cardholders will not default, which is higher than in the original data. However, we also lose 17% of customers who will not default as we classify them as positive. We are making a trade-off between having a lower default rate and having more customers who will not default.

|  |  |  |
| --- | --- | --- |
| Loss | Accuracy | AUC |
| 0.6028 | 0.7882 | 0.6998 |



**C) XGBoost**

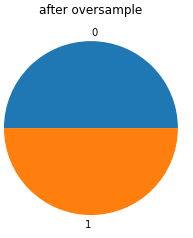
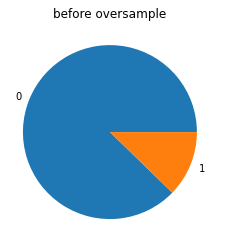
*1. Data Processing*

From the data exploration process, we can clearly see that there are two major challenges in the data that we need to solve, first is the categorical data (Applicant gender, Income\_Type), another one is the imbalanced data issue.

For categorical data, since XGBoost can’t automatically handle non-numeric data, thus we need to transfer the categorical data into numeric form. The prevailing methods of encoding categorical data are label encoding, one hot encoding, target encoding. For label encoding, it will assign a number to each possible value of a categorical variable. It's very easy to implement but with one important issue of introducing number sequencing. The rank that label encoding is introduced between the possible values of categorical data by assigning different magnitudes of numeric numbers. If our data don’t have this order, it might overestimate the importance of some value while underestimating the importance of the other. Thus label encoding is often biased as the mathematical representation is not reflective of the relationship between levels.

One hot encoding is another popular way of encoding categorical data. It avoids directly assigning a number to the possible value of categorical data. Instead, it makes the n possible value of a categorical variable into an n-1 binary variable with 1 representing true and 0 representing false. Everything sits in an orthogonal vector space. However, one hot encoding will result in high cardinality, which means that the feature space can really blow up quickly and the later curse of dimensionality issues. Besides, for non-linear machine learning models, when using one-hot encoding, it will only search for splits on single levels, which might be highly inefficient, especially when there are very many levels. Target encoding replaces features with a blend of the posterior probability of the target given a particular categorical value and the prior of the target over all the training data. It won’t introduce additional dimensions to the data. While there are still some disadvantages of target encoding, as the target encoding is dependent on the distribution of the target which means target encoding can be prone to overfitting. And also since the value of possible values depends on the label, the information leakage problem needs to be highlighted. Thus the value will only be calculated from the training set. Since the XGBoost model is used in this example and the booster is GBtree, thus it is a non-linear model. Then the target encoding method is chosen.

Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e one class label has a very high number of observations and the other has a very low number of observations. The traditional approach of classification and model accuracy calculation is not useful in the case of the imbalanced dataset. Prevailing data imbalance methods include change evaluation metric, resampling, threshold moving, and SMOTE(Synthetic Minority Oversampling Technique). Since our data is about credit card default information, the positive and negative cases are highly imbalanced due to the fact that the majority of customers will pay their credit card bills on time. Without rebalancing the data, we won’t be able to achieve a useful classification model. Here, an oversampling method is implemented.



*2. Evaluation Metrics*

As our task is binary classification, we use Binary Cross Entropy as the loss function. We also include accuracy, F1 score and AUC in our validation set for hyperparameter tuning and testing process as metrics to decide how well our models perform.

*3. Models Design and Major Results*

1. Base configuration

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It is an implementation of gradient boosting machines created by Tianqi Chen. Two reasons to choose XGBoost. The first one is the execution speed.It’s faster when compared to other implementations of gradient boosting. The second one is the overall model performance. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. The evidence is that it is the go-to algorithm for competition winners on the Kaggle competitive data science platform.

XGBoost is based on boosting ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. This approach supports both regression and classification predictive modeling problems.

b. Hyperparameter Tuning

In general, there are three types of hyperparameters including general parameters, booster parameters and task parameters.

General parameters consist of booster and number of threads. Booster helps us to choose which booster to use. Here, since it’s a classification problem, I choose the gbtree model. Number of threads control the number of parallel threads used to run XGBoost. It is used for parallel processing and the number of cores in the system should be entered. I leave blank here to let the system automatically detect.

Booster Parameters consist of eta, gamma, max\_depth, min\_child\_weight, lambda and alpha. The eta is analogous to learning rate. It is the step size shrinkage used in updates to prevent overfitting. Here, I choose the eta of 1. Gamma controls the minimum loss reduction required to make a split. Larger gamma will make a more conservative algorithm. Here, I choose default gamma, which is the 0 stands that as long as loss function has reduction, the tree will continue to grow. Max depth stands for the maximum depth of a tree. It is used to control over-fitting as higher depth will allow models to learn relations very specific to a particular sample. By cross validation grid search, optimal max\_depth in our case is 5.The min\_child\_weight defines the minimum sum of weights of all observations required in a child. By cross validation grid search, the optimal value of min\_child\_weight is 2. Lambda controls the weights of L2 regularization and alpha controls the weights of L1 regularization. In here, since our base model is a tree model, thus we don’t need to standardize our predictor, by grid search, alpha is 0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ETA | Max\_Depth | Min\_Child\_Qeight | Alpha | Booster | Gamma | N\_estimator |
| 1 | 5 | 2 | 0 | GBtree | 0 | 190 |

|  |  |
| --- | --- |
|  |  |

*4. Result interpretation and analysis*

1. Shap value

In machine learning there’s a recurrent dilemma between performance and interpretation. Usually, the better the model, the more complex and less understandable. XGBoost is not an entirely black box model but still not easy to explain like a linear model whose coefficient is the influence of a predictor to the overall result. Thus, we introduce shap value. SHAP values interpret the impact of having a certain value for a given feature in comparison to the prediction we'd make if that feature took some baseline value. It is first used to ensure equal distribution in game theory.

From the graph, we can see that the number of total good debt is the predictor with highest shap value, thus the largest contribution to the result. Also our label means if a person will default, thus the larger the total number of good debt, the less likely that a user will default. Total income is the second highest shap value, and the lowest income the highest chance of getting default. Year of working and total children has a negative relationship with the default rate, the former may due to people who work for a long time will have more stable income, latter may due to a family that can afford to raise more children may also be able to pay bills on time.

|  |  |
| --- | --- |
|  |  |

**IV. Final Results and Future Work**

In this report, we address the classification prediction of the issuing credit card by comparing the performance of testing sets among manydifferent models. Compared to all the methods we use, the XGboost method and Neural network reach relatively the same performance with AUC of 0.7 and accuracy of 0.88. But considering the model explanation ability, computation cost, model complexity, XGboost outperforms neural networks. In the future, we will cover more models, better data feature engineering procedures, and feature selection.

References

1. Seanny, 2020, Credit Card Approval Prediction, Data source retrieved from: https://www.kaggle.com/rikdifos/credit-card-approval-prediction
2. Data Repository: https://github.com/qiuke12321/CS
3. Antonie Baum, Pycaret, Pycaret Package retrieved from: https://pycaret.readthedocs.io/en/latest/api/classification.html
4. Caesarmario, ETL Credit Card Data Set, retrieved from: [https://gith](https://github.com/caesarmario/etl-credit-card-dataset-using-pentaho)ub.com/caesarmario/etl-credit-card-dataset-using-pentaho
5. Jan Majewski, Training Neural Networks for price prediction with TensorFlow,from:https://towardsdatascience.com/training-neural-networks-for-price-prediction-with-tensorflow-8aafe0c55198
6. Jason Brownlee, SMOTE for Imbalanced Classification with Python, from:https://stats.stackexchange.com/questions/282160/how-is-it-possible-that-validation-loss-is-increasing-while-validation-accuracy