PREDICTING CREDIT CARD DEFAULT

A PREPRINT

Group Name: `Group A`
Group contribution percentages: Ke Ding: 40%, Gautam Prakash:15%, Saeef Rosin:15%, Chang Lu:15%, Yassine Jouaouine:15%
Department of Biomechanical Engineering
University College London
London, WC1E 6BT

January 8, 2020

1 Introduction

As the topic of credit card default prediction which this project centers on is widely discussed, and the dataset provided is well explored, the first take to the assignment was to review research papers which tackle the similar problem with vastly diverse approaches and methodologies. After highlighting consistent methods and interesting techniques from literature we were able to begin treat the training data. We commence with visualising the data using numerous graphs to pick up on trends, patterns and any abnormalities in the data. Secondly Feature Selection is carried out which removes any data deemed unnecessary. The is done to reduce complexity and overfitting of models. With a more streamlined dataset we can start model training and validation. a total of 6 classification models will be cross-compared using multiple performance metrics.

The highest accuracy score amongst all 6 models obtained is 0.8184, from based training data (scaled). However, scores in almost all performance metrics change considerably after applying data resampling, therefore making the model evaluation and selection process not so straightforward. This implicitly demands a thorough understanding of each performance metric and how their scorings would be affected by the data engineering applied.

2 Data Exploration & Transformation

2.1 Data import and cleaning

The training data is imported into a pandas dataframe, with all data type tranformed from object into int64, to allow future manipulation through numerical data handling methods. And for that same reason, column description row is deleted, leaving only the column labels (X plus the number). To compensate and make the following explanations and analysis more readable, a description to column labels is attached below, see table 1.

2.2 Feature exploration and visualisations

Features from the imported training data contains 23 columns and 24,000 rows. To get some first impressions on how the feature data are distributed, the pandas dataframe's describe method is used, obtaining the following statistics table, see figure 1.

Also, histograms, box-and-whisker plots and kernel density plots are used to help better visualize the feature distributions (see the corresponding jupyter notebook sections for all plots for the separated classes)

It can be seen that the minimum, maximum, medium and quartiles of the data for each column has been calculated. The values in X1 show that only 25% of the customers have been given credit of more than $250,000. The interquartile

Table 1: Column description sheet

Column X label	Description to the column
X1	Amount of the given credit
X2	Gender
X3	Education
X4	Marital status
X5	Age
X6 to X11	History of past payment each month
X12 to X17	Amount of bill statement each month
X18 to X23	Amount of previous payment each month
Y	default outcome (0: No default, 1: Default)

	count	mean	std	min	25%	50%	75%	max
X1	24000.0	165495.986667	129128.744855	10000.0	50000.00	140000.0	240000.00	1000000.0
X2	24000.0	1.628250	0.483282	1.0	1.00	2.0	2.00	2.0
X3	24000.0	1.837542	0.737341	1.0	1.00	2.0	2.00	4.0
X4	24000.0	1.562375	0.521492	1.0	1.00	2.0	2.00	3.0
X5	24000.0	35.380458	9.271050	21.0	28.00	34.0	41.00	79.0
X6	24000.0	-0.003125	1.123425	-2.0	-1.00	0.0	0.00	8.0
X7	24000.0	-0.123500	1.200580	-2.0	-1.00	0.0	0.00	8.0
X8	24000.0	-0.154750	1.204033	-2.0	-1.00	0.0	0.00	8.0
X9	24000.0	-0.211667	1.166549	-2.0	-1.00	0.0	0.00	8.0
X10	24000.0	-0.252917	1.136993	-2.0	-1.00	0.0	0.00	8.0
X11	24000.0	-0.278000	1.158169	-2.0	-1.00	0.0	0.00	8.0
X12	24000.0	50596.884708	72649.374256	-165580.0	3631.50	22330.0	65779.50	964511.0
X13	24000.0	48646.064125	70364.600436	-69777.0	3098.50	21339.0	62761.25	983931.0
X14	24000.0	46367.057625	68193.898321	-157264.0	2773.50	20039.0	59298.00	1664089.0
X15	24000.0	42368.188417	63070.680934	-170000.0	2340.00	18940.5	52188.50	891586.0
X16	24000.0	40000.682542	60345.012766	-81334.0	1740.00	18107.5	49746.50	927171.0
X17	24000.0	38563.710625	59155.759799	-339603.0	1234.75	17036.0	48796.25	961664.0
X18	24000.0	5542.912917	15068.576072	0.0	1000.00	2100.0	5000.00	505000.0

Figure 1: Simple stats table for features

range of X1 is $190,000 and so outliers are the ones above $535,000. The values of X6-X11 show that up to half of the customers paid before they were due and only less than 25% of them delayed their payment by at least one month.

X3, the level of education completed for each of these people is indicated as 1 for graduate school, 2 for university, 3 for high school and 4 and above for others. It is shown that 25-50% have only completed graduate school and less than 25% have completed high school or others.The age of people, X5 shows that half of them are between 21-34 and only a quarter of them over 41.The values of X12-X17 show that less than 25% of customers have a bill statement of less than 0. The box plot of these values show that there are many outlies in the bill statement values given, that they are more than 1.5 times the interquartile range above the upper quartile. So their bill statement values are way above the range most customers have. The values X18-X23 all have many outliers as the values of 1.5 times the interquartile range above the upper quartile are all between 10000-15000 and there are values of amount paid above 400000 and for one of the X19 the highest is 1684259.

2.3 Class visualisations

From figure 2, it can be observed that it is very unbalanced as class "0" is much larger. This shows the majority of the data is comprised of 18630 non-defaulting customers, and only 5370 default cases.

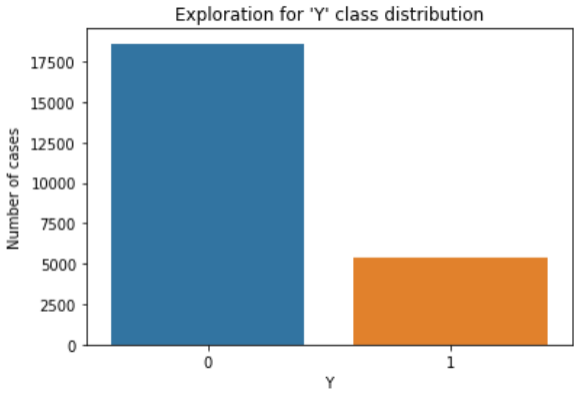

Figure 2: Bar chart showing class distributions

2.4 Data transformations

From the initial data exploration obtained from figure 1, the minimum values for data column X3 and X4 look very suspicious, as they are both labelled “0” while there is no information for type “0” for both of these columns from the data description sheet provided. Based on that observation, the dataframe’s values count method is used to reveal all the different types of labels, as well as their corresponding number. From figure 3, one can easily spot there are 11 type “0”s, 207 type “5”s and 37 type “6”s that are unknown for column X3; there are 41 type “0”s that are unknown for column X4. To deal with these unknown labels, a logical decision was made to categorise all unknowns into the type “other” for the respective columns (“4” for X3, “3” for X4), complying with the data description sheet. The unknown data are not deleted for the consideration that they might contain the usable information contributing to final classification, and a deletion would mean a complete loss of information. To achieve such re-categorisation, the apply method coupled with a lambda conditional labelling function is used to pick out all the unknown types and re-label them with the “other” type. Results shown in figure 4 demonstrate the success of such method.

```
2    11360
1     8443
3     3850
5      207
4       92
6       37
0       11
Name: X3, dtype: int64
 2    12877
1    10813
3      269
0       41
Name: X4, dtype: int64
```

Figure 3: Type distributions before correction

Transforming features is an important process for machine learning. It can improve the accuracy of the prediction. Apart from the previous re-labelling of unknown types, there are additional steps taken in this project to attempt to

```
2    11360
1     8443
3     3850
4      347
Name: X3, dtype: int64
2    12877
1    10813
3      310
Name: X4, dtype: int64
```

Figure 4: Type distributions after correction

create the most suitable data to feed into our chosen classification mode. One approach done is to take a log transform. Log transform can adjust the skewed data and normalize the data distribution. It also helps reduce the magnitude difference between different features so that magnitude order effect can be reduced. The other method applied is DBScan Clustering. In dataset, it is used to detect anomaly data on the density basis. This algorithm will cover the core points and border points, so the left will be considered as outliers. Outlier points do not belong to any cluster, they may be anomalous that cause noises. StandardScaler is also universally applied to data. It uses the principle of standard deviation, scaling down data by removing the mean and transforming to unit variance. StandardScaler is similar to log transform on some extent. Therefore, they can give similar outcomes.

2.5 Feature selection

Before selecting the features, it is necessary to have a general picture of how much weight each feature has and their overall quality. In this project, random forest classifier method provided just that to rate the significance of each features. Higher significance means it contribute more to the machine learning prediction accuracy. In general, a random forest algorithm comprises of four steps. Firstly it will select random samples in dataframe, then a decision tree will be created for each sample to predict result. As it uses voting mechanism to predict results, the final prediction will be the one received the most votes. From the figure 5, it shows X6, X5 and X1 have the highest significance, which means these features show be paid attention, while the X2 (gender) and X4 (marital status) should be deleted because their noises may contribute more to the prediction accuracy. Meanwhile, the msno matrix method from missingno is used to test the data quality (to see if there is any missing data). The nullity matrix gives a data dense display, and no white gaps appearing in the grey bars in our plot confirms that no missing data in this training set that needs to be otherwise treated. (see the corresponding section of jupyter notebook for this plot)

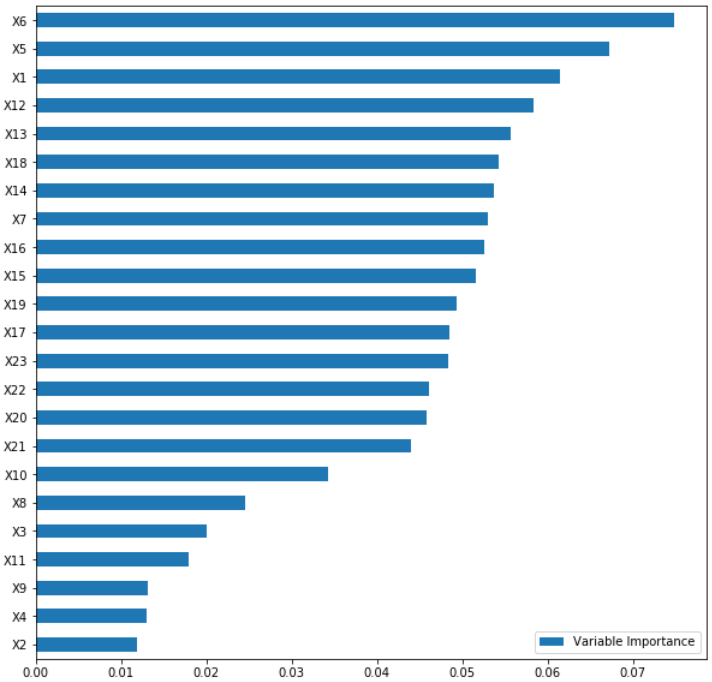

Figure 5: Feature importance ranking

To get the feature correlation relationship, a specific algorithm is created in this project for features selection. Its operating mechanism is to rating the similarity of all 22 features, and set a threshold to keep only one of those features whose correlation rate is higher than 0.8. By this algorithm, a heatmap graph was plotted (see figure 6) to give a visualization. The big red square in the middle indicates a high correlation between the corresponding columns. X9, X11, X13, X14, X15, X16 and X17 was suggested that should be deleted as they are highly correlated to X10 and X12. This process is critical, because it can condense the value of a dataset, delete prolix data and also clean the noise in case of overfit.

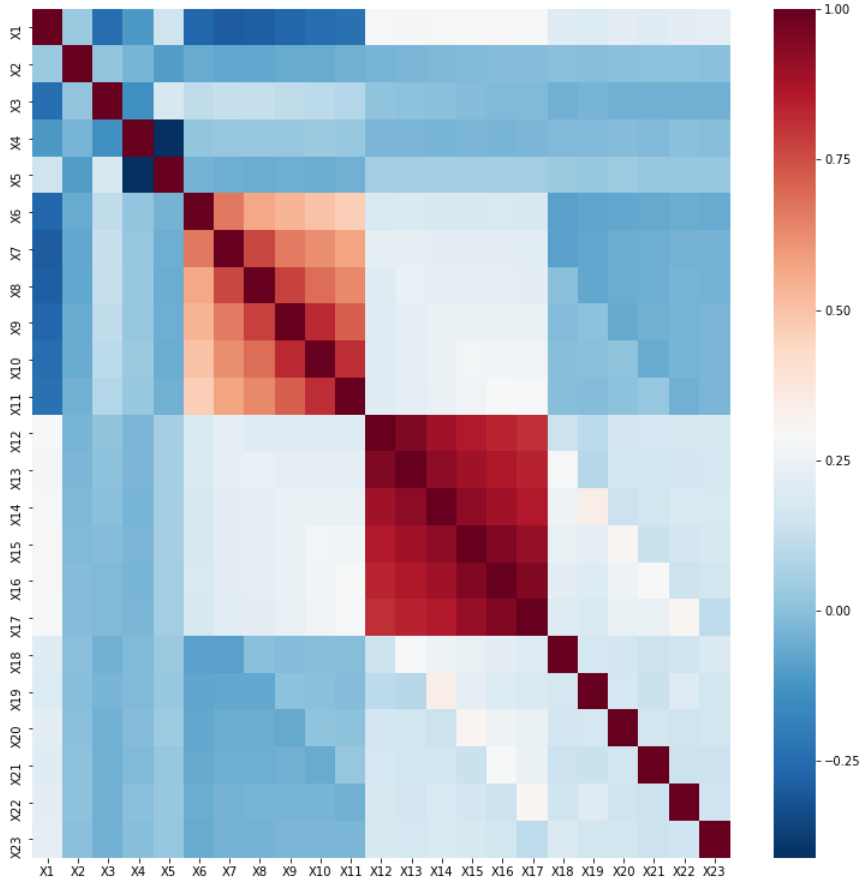

Figure 6: Feature correlation heatmap

3 Methodology overview

3.1 Background readings and literature review

A number of source material was reviewed beforehand which aimed at also predicting defaults with the same data set. From the review it was clear that each method followed consistent steps. From separating data to form estimate and validation samples. Here is where different classification methods are compared via metrics.Some examples being accuracy (overall how model predicts both defaults and non-defaults) and recall (The proportion of actual defaulters the model will predict correctly). Different papers opted to use different metrics. Some just using accuracy without further reasoning. One source [1] argues that predicting a customer to be a payer when they in fact default is least desirable from a banking point of view. So in the interest of profits it was decided by this paper that maximising recall was a priority. This same source also investigated the relationship of metrics recall and precision with classification threshold. This allowed an optimum threshold to be selected to further maximise recall.

Another source [3] stated the the ratio between non-defaulters and defaulters was too high (78:22). Running logistic regression yielded an accuracy of 80% arguing that the classifier would be no better than randomly guessing. This gives an early impression that logistic regression is not suitable for this particular dataset.

Finally, one approach[6] looked at both increasing accuracy and recall. They deemed relying on one type of classification to be insufficient. They proposed a unanimous voting with 5 classifications. This means all 5 classifications have to deem a customer a payer for the overall prediction to be a non-defaulter.

3.2 Methodology adopted

Given all the previous information for how to select the model and the basis on which a model should be assessed, it is concluded that and many models could be tried and many performance metrics should be considered to ensure, a minimised bias in the evaluation of a classification for a particular model. The approach from a defaulter catching project[10] provides particular inspiration to unfold such a multi-model multi-metric approach.

The approach that we adopt primarily employs the k-fold cross validation method from sklearn for all the evaluation. Knowledge from the Model Selection lecture provides the theoretical basis for the need of validation set as a proxy for the generalisation loss, so that the model's eventual behaviour on the test set would not be far off from that obtained through the training/validation set. Data is split into training set, k-folds of validation set, and a hold-out test set. A number of classifier models (mostly from sklearn) and performance metrics on which each model will be scored on are selected. Scaling is used for the input data for all classifier training. Feature engineering is applied to the training data and the model training is done on several different versions of the original data. The validated results are computed to form a score table, on which the performances are cross-compared. The optimal model is selected based on maximum performances in some particular metrics that are deemed more essential for the application of this project, while compromises in others. Hyperparameters of the optimal model are tuned prior to be put into use for the final test set. The final results on test set are visualised and analysed.

3.3 Initial models chosen for trial

- **Logistic Regression (from sklearn)**
  Although through literature review, it is thought that logistic regression may not be suited to an imbalanced class. But It is still selected as it can give a probabilistic interpretation of an event and handle both linear and non-linear decision boundaries.
- **Linear Support Vector Classifier (SVM from sklearn)**
  By trying to compute a hyperplane of best fit to the data, Linear SVC divides data into classes. It is selected as it tends to be less prone to overfitting, especially on high dimensionality feature space.
- **Gaussian Naïve Bayes (from sklearn)**
  Based on conditional probability, Naive Bayes is rather simple to implement. Although the conditional independence assumption that it requires is not often true in practice, it is selected for its ability to make easy and fast predictions.
- **Linear Discriminant Analysis (from sklearn)**
  LDA assumes a Gaussian dataset and equal variances. It uses Bayes theorem to probability to each class, then chooses the highest. LDA is chosen as it is well suited to make classification with categorical output variable.
- **Quadratic Discriminant Analysis (from sklearn)**
  QDA differs from LDA with each covariance matrix is computed for each class.It is selected in case that each individual class may or may not have similar covariances. Consequently, one can predict that only one of the models from LDA and QDA will perform well on this dataset.
- **XGBClassifier (from xgboost)**
  The principle of XGBClassifier underpins gradient boosting. When used for classification tasks, it uses tree booster to perform parallel tree boosting. It is selected for its high accuracy and efficiency, as it is highly optimised for classification.

3.4 Metrics:

Each method will be judged on 5 metrics listed below. Each metric can be derived from the confusion matrix. This is the matrix that compared the predicted and actual classifications from test data. For this assignment a positive is the prediction or result that a customer is a defaulter.

- **Accuracy:**
  This is the percentage of correct predictions a model makes on a test set. In our case it is the proportion of defaulters and payers the model predicts correctly.[7]

- **Precision:**
  This is the percentage of positive predictions that are correct. In other words, of the amount of customers deemed to be defaulters, how many actually are.[7]

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive+False Positive}}$$

- **Recall:**
  The percentage of actual positives that are correctly classified. In context, this is the percentage of actual defaulters that are caught out by the model. [7]

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive+False Negative}}$$

- **F1 Score:**
  Is a combination of both precision and recall. Typically used when both precision and recall are deemed equally important. [7]

$$\text{F1 Score} = 2 \times \frac{\text{PrecisionRecall}}{\text{Precision+Recall}}$$

- **ROC/AUC:**
  This metric uses true positive (recall) and false positive rates to form a graph. Multiple methods can be plotted on this graph. The classification threshold provides different coordinates on the graph. Joining the points together form the ROC curve. The area under the curve can be used to rank methods. A larger area indicates a classification is superior.

3.5 Feature engineering

As one can recall from figure 2, the class distribution is very unbalanced. It is this unbalance that prompts the use of a series of resampling techniques, to restore balance to the class data. We believe this might be helpful as many of our chosen models work well on datasets with balanced classes.

- **Synthetic Minority Over-sampling (SMOTE)**
  The first resampling applied is SMOTE. It is an over-sampling that increases the number of underrepresented class (rare cases). It generates new entries from the underrepresented cases based on corresponding or near samples from the feature space. SMOTE algorithm actually computes the distance between one minority class data point and its neighbors and locate its nearest neighbor (of the minority class), and engineer a new instance with that minority label randomly between the linear distance of those two points [11]. SMOTE is used in this project as a top choice for resampling because it creates artificial features that are close to the original features, thus having less effect on the overall feature distribution. To implement, smote from imblearn is employed as it is a very straightforward package to use.
- **Over-sampling**
  Over-sampling is applied as a naive method to raise the number of instances in the minority class. It functions by adding random samples from the original minority set to the new set. It is basically a random-selection process.
- **Under-sampling**
  Under-sampling, contrary to over-sampling, randomly removes samples from the majority class. This sometimes will cause changes in the variance, as well as a loss of important information as this method simply removes samples. Therefore it is reserved as the last option for resampling.

4 Model training and validation

First, we examine the data: the project's provided data comes already in the form of a training set, 24000 entries, and a separated 6000-entry test set. The approach assesses models with cross validation, thus implying that the 24000-entry set should be further split into training and validation sets. And again the technique of k-fold cross validation is used as it splits and utilises the data more efficiently (randomly choose one to validate and the rest act as training data). Therefore kFold from sklearn is selected for a simple implementation.

To begin the training, first, the dataframe df2 (it is the main df after feature selection) is used as the base data. Df2 is split into feature matrix (X), and target (y). A standard scaler from sklearn is applied to the feature (X) as it is observed from previous exploration that values in some feature columns are much bigger than others (i.e. have drastically different scales). Next, a custom function is declared to make the cross validation and scoring process universal for all 6 models. Within the get_cv_scores_base function, by default 5 validation folds are generated using kFold method. Then cross_val_score returns the scores for each model for each metric. Dataframe df_model then collects all the scores. The benefit of having such a general function is that it can be called for every model at any time any where in the algorithm. By concatenating all scores, a table is computed showing the results from train/cross validating all models on base data (with scaling), see figure 8.

Now since a performance benchmark for all models using the base data is available, it is appealing to go back to the data and try some feature engineering that might improve the current performance further. The first resampling we try is SMOTE. To incorporate SMOTE into the cross-validation process, a new function is declared. The get_cv_scores_smote takes in the same base data (with scaling), generate a 5-folds split, and performs a manual cross validation. From the cv.split object, each individual training and validation sets can be accessed and populated. Each model is trained with the already-split data after undergoing the fit_sample method from smote. Scores for each set stored in a list, and the list elements are simply averaged to get the validation performance. This is because of the cross validation loss which is merely an average across all k folds, according to the equation in figure 7. And the result table for this smote dataset is displayed in figure 9.

$$\mathsf{L}_{\mathsf{CV}_k}(\mathcal{E}, \mathcal{S}, f) = \frac{1}{k} \sum_{i=1}^{k} \mathsf{L}_{\mathcal{S} \setminus \mathcal{S}_k}(\mathcal{E}, \mathcal{S} \setminus \mathcal{S}_k, f)$$

Figure 7: The equation for cross validation loss

Similarly, over-sampling and under-sampling are all implemented, through similar custom functions get_cv_scores_ovr and get_cv_scores_udr. Results for those resampled datasets are displayed in figure 10 and figure 11.

5 Results

Below are the 4 table containing performances of all 5 metrics used, for all 6 models, corresponding to 4 different versions of the training data used.

	model	accuracy	precision	recall	f1score	roc/auc
0	LogisticRegression	0.807388	0.709709	0.235176	0.353170	0.718322
1	Naive Bayes	0.613285	0.335647	0.717946	0.454944	0.731319
2	Linear Discriminant	0.808452	0.696107	0.254744	0.372802	0.714319
3	Quadratic Discriminant	0.600772	0.333731	0.745180	0.457276	0.726117
4	LinearSVC	0.799983	0.726203	0.169085	0.274770	0.715453
5	XGBoost	0.818410	0.678320	0.357279	0.467943	0.775521

Figure 8: Result table for base data

	model	accuracy	precision	recall	f1	roc/auc
0	LogisticRegression	0.680539	0.375596	0.641296	0.473485	0.666677
1	Naive Bayes	0.387293	0.254624	0.899570	0.396884	0.569359
2	Linear Discriminant	0.687263	0.380990	0.631794	0.475083	0.667631
3	Quadratic Discriminant	0.360399	0.248644	0.916796	0.391169	0.558177
4	LinearSVC	0.684455	0.378721	0.635833	0.474410	0.667268
5	XGBoost	0.788885	0.530744	0.514876	0.522090	0.691564

Figure 9: Result table for data with SMOTE

From figure 8 (base data) alone, if we only consider accuracy, the cross validation shows that the best models would be either Linear regression, LinearSVC, XGBoost or LDA as their close accuracies at 0.807, 0.800, 0.818 and 0.808. Linear SVC has highest precision at 0.726. Eliminating Naive Bayes and QDA as their accuracy and precision are the least, the chosen method could either be Logistic Regression or LinearSVC as their precisions are 0.710 and 0.726,

	model	accuracy	precision	recall	f1	roc/auc
0	LogisticRegression	0.690200	0.384177	0.633225	0.477941	0.670047
1	Naive Bayes	0.408485	0.260199	0.887958	0.402322	0.578964
2	Linear Discriminant	0.697264	0.390366	0.623520	0.479866	0.671141
3	Quadratic Discriminant	0.365547	0.250017	0.915138	0.392672	0.560939
4	LinearSVC	0.694583	0.387948	0.627024	0.479005	0.670679
5	XGBoost	0.757352	0.469870	0.630642	0.538139	0.712367

Figure 10: Result table for over-sampled data

	model	accuracy	precision	recall	f1	roc/auc
0	LogisticRegression	0.690242	0.384806	0.636716	0.479355	0.671337
1	Naive Bayes	0.432741	0.266535	0.865372	0.406770	0.586568
2	Linear Discriminant	0.696966	0.390417	0.625528	0.480391	0.671696
3	Quadratic Discriminant	0.386952	0.255458	0.902363	0.397838	0.570164
4	LinearSVC	0.694540	0.388458	0.630095	0.480224	0.671759
5	XGBoost	0.757947	0.470401	0.631535	0.538946	0.713073

Figure 11: Result table for under-sampled data

while that of XGBoost and LDA are 0.678 and 0.696. However, when looking at the recall scores, the Naive Bayes and QDA, which have the lower accuracy and precision scores have the highest recall at 0.718 and 0.745. Due to this an f1score is calculated, a weighed average between the precision and recall. The best f1scores are that of Naive Bayes at 0.455, QDA at 0.457 and XGBoost at 0.468.

Considering the above analysis on results from the base data table, xgboost with the highest ranking both in accuracy and f1 scores could be selected as the optimal model. However, prior to reaching the conclusion, results from the resampled data table are also weighted into consideration. Although a clear drop in accuracy for all models, the f1score in resampled data tables for all models increase. We focus on recall as in this application it is important to correctly identify the actual default cases, i.e. "ability to catch the real defaulters". But meanwhile we are also concerned with the precision with which the non-defaulters will not be mis-classified, therefore we also try to maximise f1 scores. The ROC/AUC score is more reflective of the overall classification accuracy, so it is also considered. Based on the focused metrics above, and looking through the resampled tables, we confirm that xgboost retains a high performance in recall, f1, and roc/auc scores before and after all the resampling. Given all above, xgboost is chosen to be the optimal classifier.

6 Final predictions on the test set

Before importing the test set and doing the actual final prediction, a hyperparameter tuning for xgboost is implemented to further evaluate and refine its performance. A number of critical parameters are considered to be tuned. First of all, xgboost will be using its tree booster for classification. Among the tree booster parameter class there are some of particular importance: eta (learning rate) and max_depth. From the XGBoost official webpage, a parameter definition sheet reveals their impact on the model[9]: eta controls the feature weight shirnkage, as a mean to prevent overfitting, has a range from 0 to 1; max_depth controls how deep the tree branch would go, its increase will lead to increase in model complexity, increase the computational burden, and more likely to cause overfitting, it normally lies within 10. GridSearchCV method from sklearn is used to conduct a brute search approach to obtain the optimals of eta and max_depth. This tuning is chosen to be based on the roc/auc score, as it better represents the performance in the face of such an unbalanced dataset. The optimals are computed to be 0.05 and 3, to be passed in to the final model. This method creates a model object named gs. Finally, The trained and tuned object gs is used to predict the final test target y. Comprehensive score report is created, see figure 12. A ROC curve is also plotted, see figure 13.

7 Conclusion

Interestingly, accuracy score on final test set is 0.820 which is even higher than that on the training set (0.818). The main diagonal of the confusion matrix is 4642 and 279, which is indicative of the fact that this model has good classification ability on identifying the non-defaulters correctly, but not so much on catching the defaulters. One of the underpinning reason for this phenomenon may be because given such class imbalance, the model obtains its high precision just by selecting the more common class. For future efforts, new techniques of data engineering (create new features based on the old) may help achieve a better balance between precision and recall.

```
Accuracy Score for Optimized xgb: 0.8201666666666667
Precision Score for Optimized xgb: 0.7520215633423181
Recall Score for Optimized xgb: 0.22037914691943128
ROC_AUC score for Optimized xgb: 0.600472632183839

------------------------------------------------------------
Classification report for Optimized xgb:
              precision    recall  f1-score   support

           0       0.82      0.98      0.90      4734
           1       0.75      0.22      0.34      1266

    accuracy                           0.82      6000
   macro avg       0.79      0.60      0.62      6000
weighted avg       0.81      0.82      0.78      6000

Confusion Matrix for Optimized xgb:
 [[4642   92]
 [ 987  279]]
```

Figure 12: Final classification scores and report

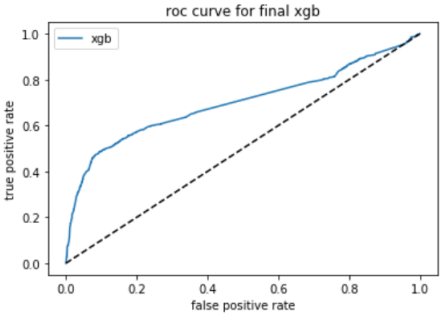

Figure 13: ROC curve for final predictions

References

[1] A. Fuentes, Hands-On Predictive Analytics with Python: Master the complete predictive analytics process, from problem definition to model deployment Packt Publishing Ltd, 2017.

[2] A.Husejinovic, D.Kečo and Z.Mašetić, "Application of Machine Learning Algorithms in Credit Card Default Payment Prediction," International Journal of Scientific Research, 2018.

[3] V.Waingankar, "Predict Default of Credit Card Clients," Zenodo.

[4] S.R.Islam, W.Eberle and S.K.Ghafoor, "Credit Default Mining Using Combined Machine Learning and Heuristic Approach," Tennessee Technological University.

[5] T.Evgeniou and S.Zoumpoulis. "Classification for Credit Card Default," [Online]. Available at: http://inseaddataanalytics.github.io/INSEADAnalytics/CourseSessions/ClassificationProcessCreditCardDefault.html. [Accessed 27 12 2019]

[6] D.Liang, C-F.Tsai, A-J.Dai, W.Eberle "A novel classifier ensemble approach for financial distress prediction," Springer-Verlag, London, 2017.

[7] K. P. Shung, "Accuracy, Precision, Recall or F1?," towards data science, 25 March 2018. [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. [Accessed 29 12 2019].

[8] I-C.Yeh, C-H.Lien, "The comparisons of data mining techniques for the predictive," Elsevier, 2007.

[9] "XGBoost Parameters," XGBoost, [Online]. Available: https://xgboost.readthedocs.io/en/latest/parameter.html. [Accessed 06 01 20].

[10] X.J.Seow, "Catching a Welcher: Classifying a Credit Card Defaulter", towards data science, 19 Nov 2019. [Online]. Available: https://towardsdatascience.com/catching-a-welcher-classifying-a-credit-card-defaulter-f4b21547a618. [Accessed 01 01 20]

[11] J.D.Dios Santos "Handling Imbalanced Datasets with SMOTE in Python", kite, 21 Aug 2019, [Online].Available:https://kite.com/blog/python/smote-python-imbalanced-learn-for-oversampling/ [Accessed 06 01 20]