Credit Card Default Predictor

Ruoting Liang, Lichen Ni, Leiling Tao, Yuyi Zhou
Department of Computing Science
Simon Fraser University

Abstract

In this report, we constructed a machine learning pipeline for a credit card default classifier using data from UCI Machine Learning Repository. After extensive exploratory data analysis and feature engineering, we compared the performance of 8 different models, including logistic regression, decision tree, Naive Bayes classifier, random forest, extreme gradient boosting, support vector machines, multilayer perceptron and deep learning on the dataset. Performance of each model was evaluated using precision, recall and F1 score. Finally, we ensembled all models using majority count. Our best single model was random forest, while the ensembled model achieved the highest F1 score.

1 Overview

Since the last decade, machine learning has been playing an important role in various financial services, such as algorithmic trading, portfolio management and fraud detection. A less explored area, is whether machine learning can be used to predict credit card defaults. Each year, Canadian banks lose millions of dollars due to credit card delinquencies. Successful predictions of credit card default can help with early detection of credit delinquency and reduce potential financial losses.

The data from UCI Machine Learning Repository contains 30000 credit card records, which includes demographic information and payment history of their holders. Predicting whether a credit card will default or not is a supervised learning task. In the sections below, we presented our exploratory data analysis, where we trained and compared the performance of 8 machine learning algorithms on the dataset. We also ensembled all models using majority count. Finally, we discussed feature importance from the best model.

2 Exploratory Data Analysis

Exploratory data analysis (EDA) is the first step in our machine learning pipeline. As it is difficult to examine the whole dataset to determine important characteristics of the data, EDA [**EDA**] employs a variety of techniques (mostly graphical) to maximize insight into a dataset, uncover underlying structure, extract important variables, detect outliers and anomalies, and test underlying assumptions.

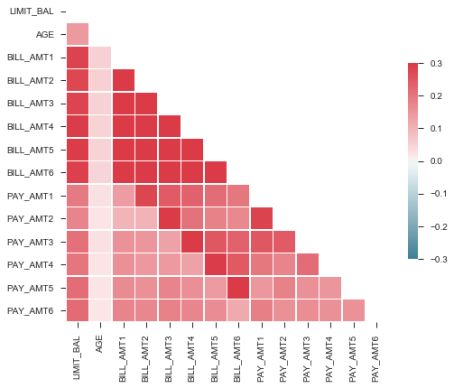
2.1 Multicollinearity

Identifying highly correlated predictors may allow us to find predictors which are proxies for another variable, or alternatively suggest variables which may be combined (e.g. via PCA). Although machine learning methods have usually been designed to be robust in the presence of correlated predictors, understanding the degree of correlation is often a useful step in producing a robust and accurate model, and is a useful aid for obtaining an optimized model.

Figure 1 shows that strong multicollinearity exists among all the continuous variables except age, which proposes the possibility to combine all the BILL_AMT (amount of bill statement) or

PAY_AMT (amount of previous payment) features into separate two features to achieve better performance for models that are sensitive to dependence among features such as Naive Bayes.

Figure 1: Multicollinearity



2.2 Potential important features

In cases where there are hundreds of features present in the dataset, which is very common in real-life machine learning tasks, selecting important features by plotting and visualization is essential to eliminate redundant features and improve model performance.

Figure 2 and 3 show that some of the features such as LIMIT_BAL (credit limit), PAY_AMT, EDUCATION, MARRIAGE, PAY (repayment status) are clearly more important than other features such as BILL_AMT and SEX.

Figure 2: Possibly important continuous features

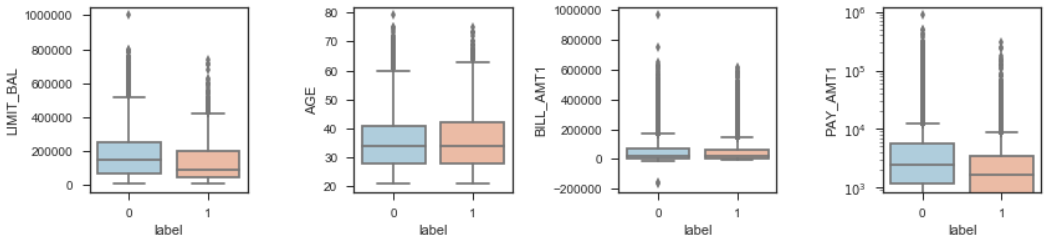
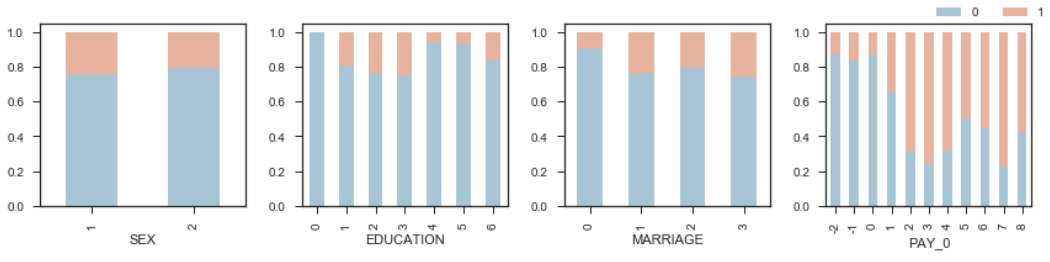


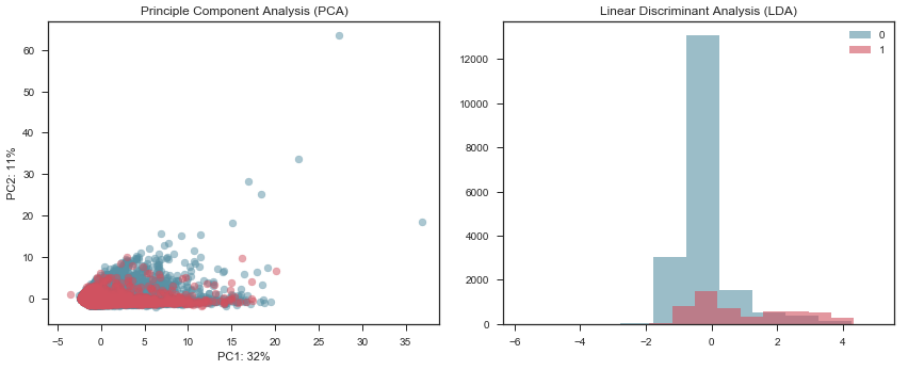
Figure 3: Possibly important categorical features



2.3 Dimensionality Reduction

Similar to feature selection, different feature extraction techniques can be employed to reduce the number of features in a dataset. The difference between feature selection and feature extraction is that while we feature selection maintains the original features, feature extraction transforms or

Figure 4: Visualizing separability of classes through PCA and LDA



projects the data onto a new feature space. In practice, feature extraction is not only used to improve storage space or the computational efficiency of the learning algorithm, but can also improve the predictive performance by reducing the curse of dimensionality.

Principle component analysis (PCA) [Shl14] is a method for unsupervised data compression, which can identify patterns in data based on their correlations. In essence, PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with fewer dimensions. On the other hand, Linear Discriminant Analysis (LDA) [MK01] is a supervised dimensionality reduction technique for maximizing class separability.

We obtained a scatter plot for the first two PCA components, which explained 43% of the total variance. For LDA, as the number of linear discriminants is at most $c - 1$, where c is the number of class labels, we plotted a histogram instead. Initial investigation from dimensionality reduction and visualization has shown that the two classes are not linearly separable, which may pose a challenge for us to construct an ideal classifier.

3 Dealing with Class Imbalance

Our data is highly unbalanced, where 78% of the samples are non-default. Thus, when we fit classifiers on such unbalanced datasets, it would make sense to focus on other metrics than accuracy when comparing different models, such as precision and recall. We decided to use F1 score as the metric to evaluate models, because our priority is to identify clients who tend to default to reduce potential financial losses (maximizing recall), without losing potential non-default clients (without losing so much precision) and F1 score combines both recall and precision:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

For all models except Naive Bayes classifier and MLP, we also tried to assign a larger penalty to incorrect predictions on the minority class, or upsample the minority class and downsample the majority class. In general, assigning a larger penalty achieved better performance compared to upsampling and downsampling, and our final models are evaluated using this method.

4 Model Comparisons

4.1 Logistic Regression

Logistic regression is a powerful linear model classifier for classification. Logistic regression model computes a weighted sum of the input features and outputs the logit of this result which is a number between 0 and 1. If the estimated probability is greater than 50%, then the model predicts the specified instance to be positive.[Gér17].

The Logistic regression model we used in Scikit-Learn library solves the following optimization problem with logit loss function and uses L2 regulation:[Lin17].

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^{l} \log(1 + e^{y_i w^T x_i})$$

To train the model, we cross-validated over a grid of parameters: C(Inverse of regularization strength):{0.001,0.01,0.1,1,10,100}, m(maximum iteration):{3,5,10}. The parameters that gave the best F1 score are C=1, m=3. It produced a F1 score of 0.5394, recall score of 0.5826, precision score of 0.5022, and accuracy score of 0.7783.

4.2 Decision Tree

Decision Tree is a non-parametric supervised learning method used for classification and regression [RN10]. It can predict the value of a target variable by learning simple decision rules inferred from the data features.

Decision tree is trained by recursively finding the most informative feature to split the data, until a stopping-condition is met, resulting in a leaf node. The predicted label is then the majority class at the node. Stopping-conditions typically include the maximum tree depth, the number of instances in each node, or whether the node can be split further. Common splitting criteria includes Gini index, entropy, and classification error. In this report we used Gini index as our splitting criterion.

$$Gini(t) = 1 - \sum_j [p(j|t)]^2$$

where $p(j|t)$ is the relative frequency of class j at note t.

To train our decision tree, in Scikit-Learn we used cross-validation over a grid of parameters: tree maximum depth: {2, 4, 6, 8, 10}, minimum number of instances per node {1, 5, 10, 15, 20}. Maximum tree depth and number of instances per node determines tree complexity, which strongly influence model fits. Our best decision tree's maximum depth is 5 and minimum samples per node is 10, which achieved an F1 score of 0.5110, precision score of 0.4349 and recall score of 0.6193, accuracy score of 0.7358.

4.3 Naive Bayes

The Naive Bayes algorithm is based on Bayes theorem with the naive assumption of independence among features. In this data set, there are both category and continuous data types. The continuous features were fitted using Gaussian Naive Bayes algorithm, while the category features were fitted using multinomial Naive Bayes algorithm in Scikit-Learn. The results were combined by multiplying predicted probability from both models. This is a simple probability classifier and does not require tuning of hyper-parameters. This model produced F1 score of 0.5256, recall score of 0.5602, precision score of 0.495, and accuracy score of 0.7747.

4.4 Random Forest

A random forest classifier is an meta-estimator that fits many decision trees to sub-samples using random features of the dataset [Ho95]. After training, all trees are averaged to make predictions. Random forest can decreases the variance of the model without increasing the bias, thus improve model-fit and reduce overfitting.

In Scikit-Learn, for each tree, we first used bootstrapping to randomly retrieve samples with replacement, then randomly chose square root of total features as maximum number of features to construct each tree. Similar to decision trees, key parameters of the model were maximum depth of each tree { 4, 6, 8, 10, 12}, minimum number of instances per node {1, 5, 10, 15}. Additionally, we also considered different numbers of trees {40, 60, 80, 100, 120}. The model that had the highest F1 score (0.5434) had an ensemble of 80 trees. Each tree had maximum depth of 10 and minimum number of instances 5. The precision, recall and accuracy scores were 0.4792, 0.6275 and 0.765, respectively.

4.5 Extreme gradient boosting

Similar to random forest, extreme gradient boosting is an ensemble of decision trees. However, instead of averaging the trees, at each step the algorithm greedily search for the best fit for the residuals from previous trees [Fri01].

$$obj^t = \sum_{i=1}^{N} LOSS(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \lambda(f_t)$$

where $\hat{y}_i^{t-1}$ is the predicted label from previous iterations, $f_t(x_i)$ is the tree at current iteration, and $\lambda(f_t)$ is the regularization component. The next best tree was found using gradient descent.

We used XGBOOST library to train the models. To find the best parameters, we varied the maximum depth of each tree {1, 4, 7}, learning rate {0.01, 0.05, 0.1} and numbers of iterations {120, 140,160,180}. The model that had the highest F1 score (0.5340) had 140 iterations, each tree had maximum depth of 4 and the learning rate was 0.05. The precision, recall and accuracy scores were 0.4457, 0.6783, 0.7403, respectively.

4.6 Support Vector Machine (SVM)

Our goal is to produce a classifier by finding the optimized hyper-plane whose boundary maximizes the margin between two classes. The distance to decision boundary is

$$\frac{t_n * y(x_n)}{|w|}$$

To maximize margin, w,b is chosen from:

$$\arg\max_{w,b}\{\frac{1}{|w|}\min_n[t_n(w^T\phi(x_n)+b)]\}$$

We can approach this optimization problem using canonical representation and constrained the optimization problem into:

$$\arg\min_{w,b}\frac{1}{2}|w|^2, s.t. \forall n, t_n(w^T\phi(x_n)+b) \geq 1$$

The lagrangian (with N multiplier a_n) is :

$$L(w,b,a) = \frac{|w|^2}{2} - \sum_{n=1}^{N} a_n\{t_n(w^t\phi(x_n)+b)-1\}$$

By a suitable choice of kernel functions, we can map the input into a higher dimensional feature space and separate it using hyper-plane. We used linear, radial basis function(Gaussian), polynomial and sigmoid kernels to train the training data. After tuning the hyper-parameters, the highest F1 score, which is obtained by using linear kernel is 0.4996.The precision, recall and accuracy scores were 0.3949, 0.6799, 0.6965, respectively.

4.7 Multilayer Perceptron (MLP)

MLP is a type of supervised learning that implements feed forward neural network model and trains using Backpropagation. Between the input and output layers, there can be one more non-linear hidden layers.[Doc17]

By using scikit learn MLP library, the model was trained using stochastic gradient descent, tuned with several hyper parameters by cross-validation. One disadvantage with MLP is that MLP with hidden layers have a non-convex loss function where exists more than one local minimum.[Doc17]. Therefore, some of the hyper parameters combinations can not converge. Results were also compared between the Sigmoid activation function and Relu activation functions in hidden layers.

By using Sigmoid activation function in the hidden layers, the parameters that produced the best result are hidden_layer_sizes=(50,50,50), max_iter=50, alpha=0.0001, learning_rate_init=0.1. The

Table 1: Model comparisons

Algorithms	F1 score	Recall	Precision
Logistic Regression	0.5394	0.5826	0.5022
Decision Tree	0.5110	0.6193	0.4349
Naive Bayes	0.5256	0.5602	0.4950
Random Forest	0.5434	0.6275	0.4792
Extreme Gradient Boosting	0.5380	0.6783	0.4457
Support Vector Machine	0.4996	0.3949	0.6799
Multilayer Perceptron	0.4915	0.3889	0.6675
Deep Learning(ANN)	0.5348	0.5924	0.4874
Ensemble	0.5467	0.5797	0.5174

model produced a F1 score of 0.4820, recall score of 0.3755, precision score of 0.6729, and accuracy score of 0.8202.

By using Relu activation function in the hidden layers, the parameters that produced the best result are hidden_layer_sizes=(50,50,50), max_iter=50, alpha=0.0001, learning_rate_init=1. The model produced a F1 score of 0.4915, recall score of 0.3889, precision score of 0.6675, and accuracy score of 0.8207.

Overall, MLP did not perform well on this data set. Relu was relatively better than the Sigmoid activation function.

4.8 Deep Learning(Artificial Neural Network)

The Keras library was used to build a deep learning ANN model with up to 15 different hidden layers on the data set. The more layers were used, the higher recall score was achieved, with the highest recall score equaling to 1. But as the recall score improved, the F1 and accuracy score declined, with the lowest accuracy score equaling to 0.2 and lowest F1 score equaling to 0.3. Additionally, when the Relu activation function was used, the results were unstable such that varying results were produced with each fit. Alternatively, the Sigmoid activation function produced more stable results than the Relu activation function.

Since the deep learning model, which requires more than 2 hidden layers, did not produce a significantly higher and more stable F1 score on this data set, a simple neural network model was chosen with 2 fully connected network layers with 30 nodes in the each layer. Each layer used Sigmoid activation function. The data set was fitted with the following parameters:[optimizer='adam', loss='binary_crossentropy', batch_size=25, epochs=5]. The differences between this model and MLP described above is that this model can assign weights to each class so that it can achieve higher F1 and recall score.

This model produced a F1 score of 0.5348, recall score of 0.5924, precision score of 0.4874, and accuracy score of 0.7703.

4.9 Ensemble

For this data set, the ensemble method was performed by using the majority vote method among all the models that were trained. Predicted labels from all models were inputted into a single matrix called all_predicts where numpy was used to produce new labels based on the majority vote.

4.10 Model comparisons

Table 1 summarized the best model from each algorithm:

5 Discussion and Conclusions

To deal with imbalanced classes of default and non-default credit card clients, we focused on reporting F1 scores rather than accuracy [YL09]. During training, we assigned weights to samples based on their frequency. In addition, we tried to up-sample the minority class and down-sample the majority class before training a model. We found that learning using weight assigning and using sampling techniques have both significantly improved model performance, while using different weights outperformed resampling methods.

From experimenting with various models on the same training and testing dataset, we found that performance of random forest classifier is superior to other models based on F1-score. In addition, random forest is computationally more efficient than many other models, such as extreme gradient boosting, MLP, SVM and deep learning. From the random forest model, the most important features are payment history and credit limit. This suggests rather than waiting for a client to miss six months of payment before defaulting his credit card (which is most banks' policy), banks can potentially reduce losses by reducing the grace period to detect problematic customers early.

After training all models, we performed ensemble learning based on majority voting, which further increased model performance. With the effect of combining several models, ensemble learning proves to produce better results in our case. However, having complex ensemble models will incur higher computation costs.

6 Contributions

All proposed the study and designed the machine learning pipelines. K.N. performed exploratory data analysis, created visualizations and trained the deep learning model. R.L. trained the SVM model and designed the report outline. L.T. trained the decision tree, random forest and extreme gradient boosting models. Y.Z. trained the NB classifier, logistic regression, MLP, deep learning and the ensemble models. All designed and presented the poster, and wrote the report.

References

[Ho95] Tin Kam Ho. "Random Decision Forests". In: *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC* (1995), pp. 278–282.

[Fri01] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine". In: *The Annals of Statistics* 29 (2001), pp. 1189–1232.

[MK01] Aleix M. Martinez and Avinash C. Kak. "PCA versus LDA". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23 (2001), pp. 228–233.

[YL09] I. C. Yeh and C. H. Lien. "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients". In: *Expert Systems with Applications* 36 (2009), pp. 2473–2480.

[RN10] Stuart Russell and Peter Norvig. In: *Artificial Intelligence: A Modern Approach* (2010).

[Shl14] Jonathon Shlens. "A Tutorial on Principal Component Analysis". In: *CoRR* abs/1404.1100 (2014). arXiv: 1404.1100. URL: http://arxiv.org/abs/1404.1100.

[Doc17] Scikit Learn Official Document. In: *Neural network models (supervised)* (2017). URL: http://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[Gér17] Aurélien Géron. In: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (2017), pp. 134–139.

[Lin17] Rong En Fan. Kai Wei Chang. Cho Jui Hsieh. Xiang Rui Wang. Chih Jen Lin. In: *LIBLINEAR: A Library for Large Linear Classification* (2017). URL: https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf.