Contents

Credit Card Default Prediction Project Report

Cici Du, David Wang, Rohit Rawat, James Kim

Summary and Introduction

Executive Summary

Here we are building a classification model using various classifiers including RBF SVM, Random Forest and Logistic Regression to predict whether the customer will default on the credit card. Our chosen classifier, Logistic Regression, performed well on the test set, with the ROC AUC score of 0.768. However, as the stronger emphasis is on correctly identifying the default class, it is alarming to see relatively low scores on both f1 and recall metrics across all the classifiers tested. It is therefore recommended to further improve this model, following the suggestions that are noted in the latter portion of this report.

Introduction

Research Question

Credit cards are now an extremely common means of transaction that most adult consumers possess these days. It is therefore very important for credit card issuing companies to be able to predict and work with the possibilities of their customers not being able to make their default payments. With this in mind, the research question that we aim to answer is: given characteristics (gender, education, age, marriage) and payment history of a customer, is he or she likely to default on the credit card payment next month?

Data

The data set that we used was put together by I-Cheng Yeh at the Department of Information Management, Chung Hua University, in Taiwan. The data set itself was sourced from the UCI Machine Learning Repository and can be found here https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Each row in the data set represents variables associated with a customer and his or her credit card payment information, including a boolean value of default. There are 30,000 observations in the data set and 23 features. There are no observations with missing values or duplicated rows in the data set.

Initial EDA

Distribution of target variables

We explored the distribution of the target variables and spotted class imbalance. Our training data contained only 22.3% of class 1 (default) in the target variable. We decided to balance the class during model training by setting class_weight to 'balanced.'

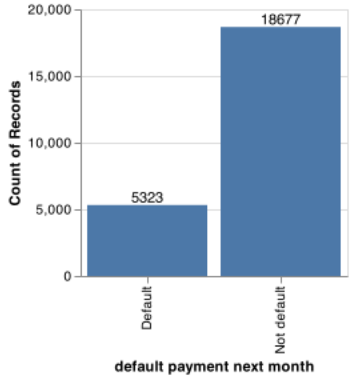

Fig. 1 Distribution of targets

Distribution of numeric and categorical features by target variable

We had 22 features and we wanted to see if any feature contributed significantly to the classification of the target variable to the extent that we could see it by plotting the distribution of each feature by the target class. We plotted the distribution of each numeric and categorical feature from the training set and colored the distribution by class (default: blue, not default: orange). We saw that the distributions below overlapped for the two classes and they looked quite similar in a lot of cases.

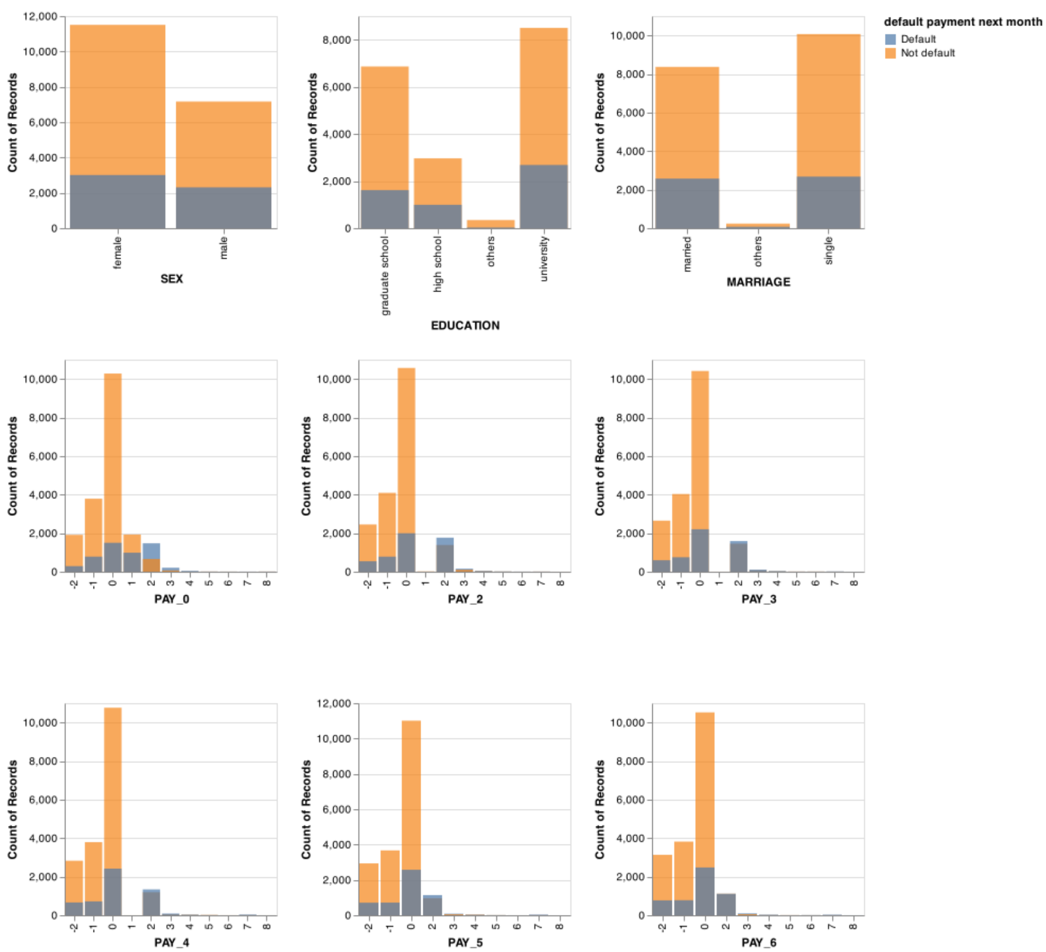

Fig. 2 Distribution of categorical features

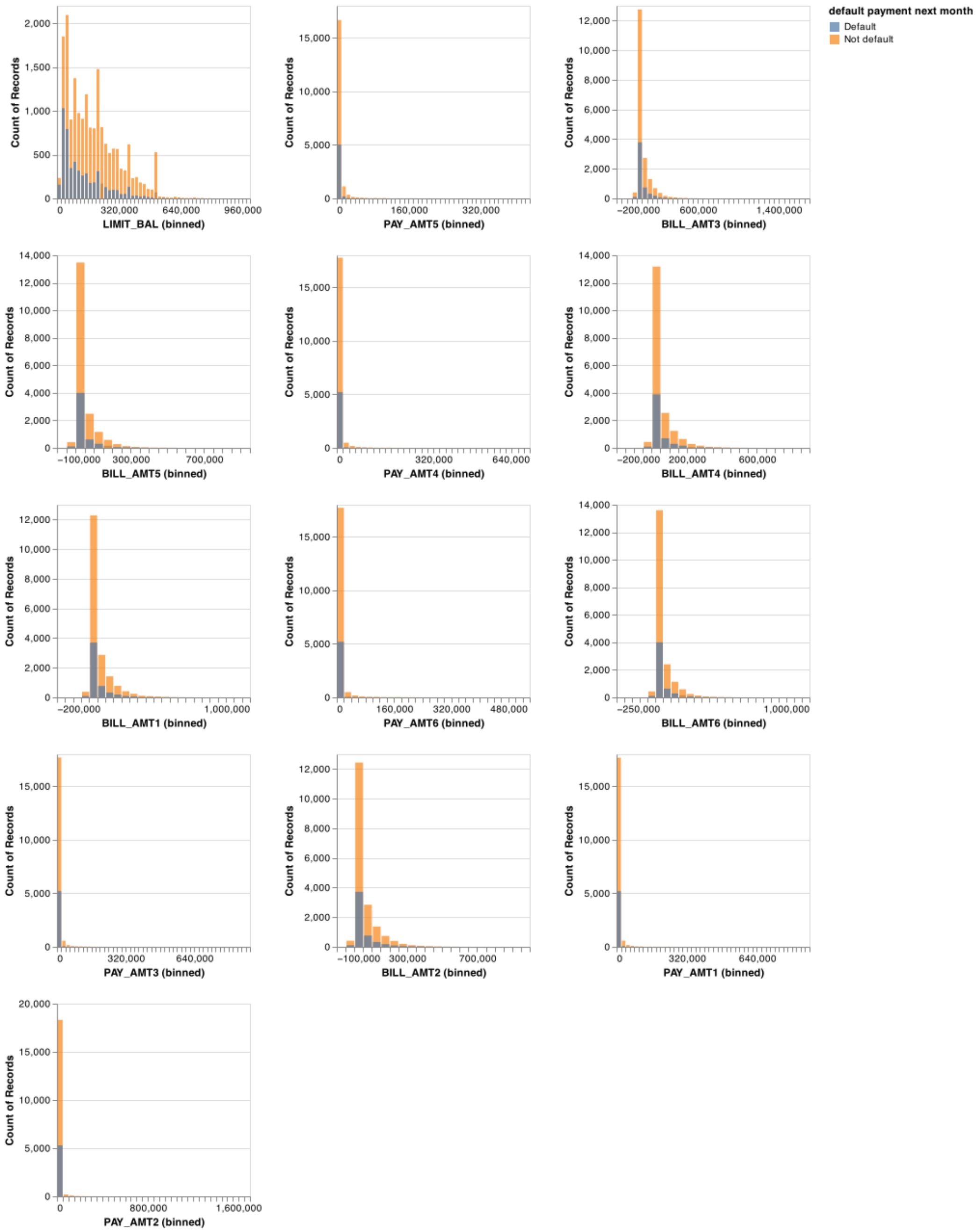

Fig. 3 Distribution of numerical features

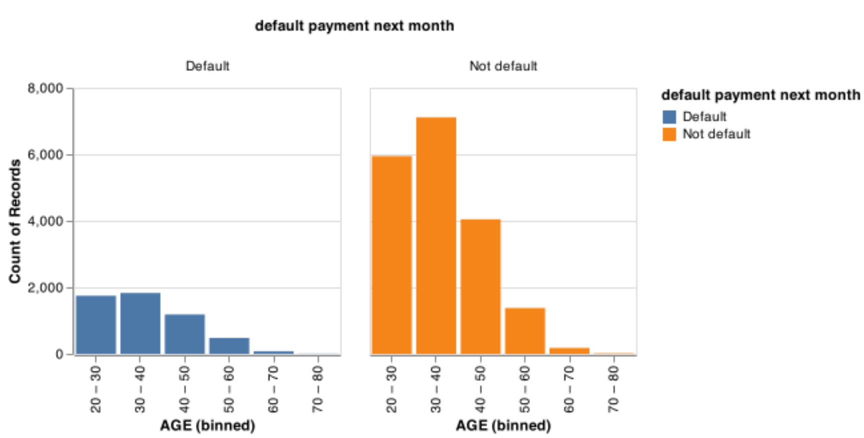

Fig. 4 Distribution of age by target

Analysis

Splitting and cleaning the data

We split our data into train and test data frames with the default setting of 0.2 split ratio. We then converted the categorical features to contain more meaningful strings as their values and the outcome file is saved in the data folder as train_visual.csv file.

Preprocessing

Since our data was relatively clean, we applied Standard Scaling on the numeric features and One Hot Encoding on the categorical features.

Choosing the best model

We trained and cross-validated the training dataset on Decision Tree, SVC, Random Forest and Logistic Regression. We also utilized the class_weight parameter and set it as ‘balanced’ to deal with the class imbalance that was observed during the initial EDA. According to our model training, Logistic Regression gave the best validation scores using ROC_AUC as the scoring method.

	Dummy Classifier	Decision Tree	RBF SVM	Random Forest	Logistic Regression
fit_time	0.019672	0.337715	28.208219	2.499096	0.436846
score_time	0.014145	0.013655	19.141284	0.161105	0.013216
test_accuracy	0.778208	0.731333	0.775125	0.816167	0.774875
train_accuracy	0.778208	0.999104	0.788677	0.999135	0.777323
test_f1	0.000000	0.394707	0.530192	0.458518	0.530696
train_f1	0.000000	0.997985	0.563065	0.998053	0.535623
test_recall	0.000000	0.395076	0.572045	0.351114	0.573928
train_recall	0.000000	1.000000	0.613940	0.999296	0.578997
test_precision	0.000000	0.394415	0.494337	0.662290	0.493733
train_precision	0.000000	0.995977	0.520053	0.996814	0.498320
test_roc_auc	0.500000	0.611161	0.764013	0.762515	0.767292
train_roc_auc	0.500000	0.999998	0.812241	0.999941	0.772885

Fig. 5 Validation scores of different classification models

Hypertuning the model

On our selected model, we tuned the parameters class_weight and C of the Logistic Regression. We obtained our best parameters and the best model which is saved as the pickle file.

C: float, default=1.0
The inverse of regularization strength; must be a positive float.

class_weight: dict or ‘balanced’, default=None
Weights associated with classes in the form {class_label: weight}.

Results

Model Results

First, here is the list of top ten features in terms of regression coefficient magnitude. These are the features that have the most impact on deciding the target class in our selected model.

	features	coefficients	magnitude
0	PAY_0_2	1.103524	1.103524
1	PAY_0_0	-1.030558	1.030558
2	PAY_0_-2	-0.854425	0.854425
3	PAY_0_3	0.733951	0.733951
4	EDUCATION_5	-0.562643	0.562643
5	PAY_0_4	0.365872	0.365872
6	PAY_0_-1	-0.360723	0.360723
7	PAY_6_0	-0.339893	0.339893
8	EDUCATION_2	0.329965	0.329965
9	MARRIAGE_0	-0.316541	0.316541

Fig. 6 Feature coefficients

We evaluated the model from pickle on the test dataset and we obtained comparable test scores to the validation score. We begin our analysis by looking at the classification report provided below.

	precision	recall	f1-score	support
No-churn	0.875648	0.829315	0.851852	4687.000000
Churn	0.487508	0.579589	0.529576	1313.000000
accuracy	0.774667	0.774667	0.774667	0.774667
macro avg	0.681578	0.704452	0.690714	6000.000000
weighted avg	0.790710	0.774667	0.781327	6000.000000

Fig. 7 Classification report of the predictions

We observe that the recall for the weighted average is better, with a higher F-1 score as well. This would mean that even though we have a class imbalance, the model performs fairly well for both categories. Another way to visualize this is by looking at the Confusion Matrix plotted below.

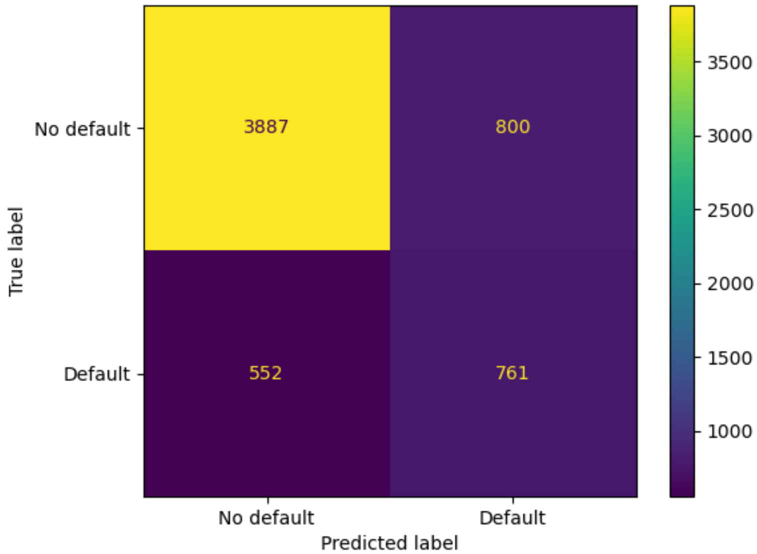

Fig. 8 Confusion Matrix of the predictions

The limitation with these metrics is that they are evaluated at the default threshold. To get a better estimate of our model performance we look at two metrics: Average Precision Score and ROC-AUC score which can be visualized in the plots shown below.

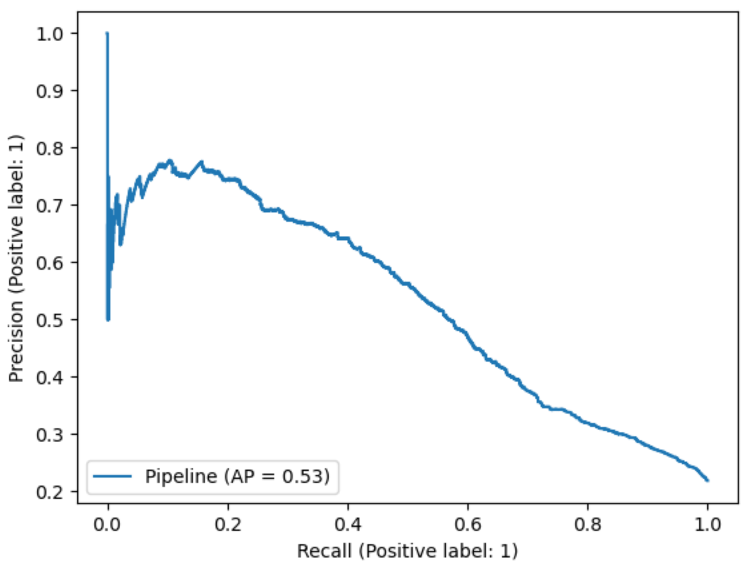

Fig. 9 Precision-Recall plot of the predictions

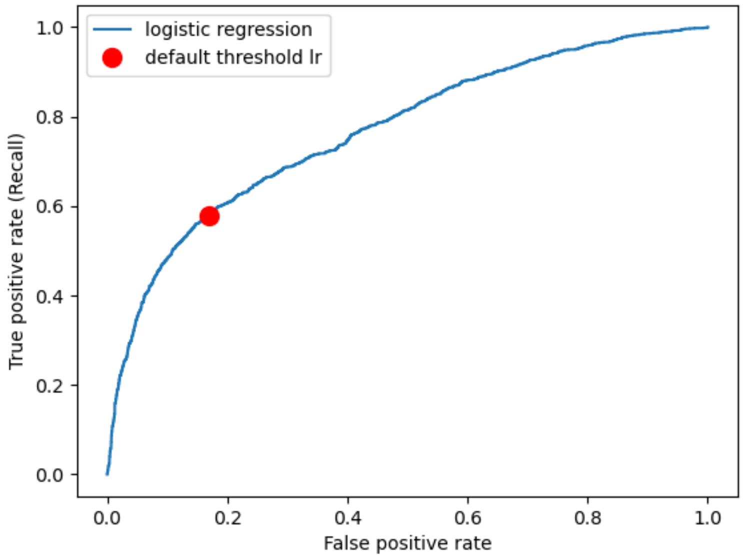

Fig. 10 ROC-AUC curve of the predictions

Since the Average Precision score is used for models having severe class imbalance, we use the ROC-AUC value to assess our final model. We are choosing ROC-AUC score over the F-1 score because we prefer a score that works well across different thresholds of the decision boundary. The average scores given by these metrics, along with other relevant scores are shown below.

	Test Scores
Accuracy	0.774667
F1	0.530000
Recall	0.580000
Precision	0.488000
ROC AUC	0.768000
Average Precision	0.529000

Fig. 11 Final scoring metrics of the predictions

Reservations and Suggestions

The major limitation of this project is that the data was collected in 2005. Consumers' spending behaviours and tastes must have changed since then so the results of this project should not be taken for granted and be blindly applied to the current setting. To further improve this model in the future, we suggest including more features such as income, vocation, size of the household, and debt to asset ratio. With more relevant features to base the predictions on, we should be able to predict our target class with more accuracy.

References

Dua, D., & Graff, C. (2017). UCI Machine Learning Repository. Opgehaal van http://archive.ics.uci.edu/ml

Python Core Team. (2019). Python: A dynamic, open source programming language. Opgehaal van Python Software Foundation website: https://www.python.org/

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in science & engineering, 9(3), 90.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Reds), Proceedings of the 9th Python in Science Conference (bll 51–56).

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. doi:10.1038/s41586-020-2649-2
