CIS 520 Fall 2019, Project Report

Risk Analysis for Lending Club Loan Data

Team Members:

- Yixue Feng; Email: `wendyfyx@seas.upenn.edu`
- Xinyu Liao; Email: `xinyul@sas.upenn.edu`
- Yebiao Jin; Email: `yebiao@sas.upenn.edu`

Abstract

In this project, we are going to analyze the default risk for Lending Club Loan data using machine learning methods. The goal is to distinguish between good loan (no default) and bad loan (default) as accurately as possible, which falls into binary classification problem. We explored multiple feature analysis and engineering techniques, binary classifier models. After comparison, our best model yields an accuracy above 70% if we consider both true positives and true negatives of equal importance on a balanced dataset which is better than most work found on Kaggle.

1 Motivation

Lending Club assigns a grade to each loan applicant by evaluating their credit risk. Then these loans can be securitized to be a financial instrument sold to investors. If a loan is of low grade (high risk), then it should require high interest rate and vice versa. So it's important to banks as well as investors to evaluate the loan quality and decide upon the credit grade of the loan as the first step of investment. Machine learning models can be exploited to analyze these problems and help banks and investors make good decisions for two reasons:

- Our task is a binary classification problem, which is actually a basic machine learning problem with many potential models available to experiment with. However the decision making process is asymmetric, e.g. banks are more afraid and will bear more loss if they predict a bad loan as good (false positives) compared to predicting a good loan as bad (false negatives). The flexibility of loss function in machine learning can address such problem and the evaluation methods such as confusion matrix and ROC can provide us with more insight.
- The abundance of loan data from Lending Club ensures that we can enough for training, validation and testing. We attempt to extract valuable insights from the dataset starting from feature selection despite the noise in the data.

2 Related Work

There are some previous work found on Kaggle with respect to this particular dataset that could help us better understand the data. Most work we found used with a highly imbalanced dataset without down-sampling, where over 90% of the data entries are "good" loans. Although they report an accuracy of 99%, because this is a binary classification task, these results are very misleading.

One experiment, which we found helpful, selected borrowers of grade E and used CatBoost, a gradient boosting library based on decision trees that is very efficient for datasets that contain categorical features with many categories, to classify their loan status (good vs. bad) [2].:

- They provided a guide on dealing with the dataset like how to deal with missing data or various data types and some insights on selecting features based on their importance.
- They shared their insights and concerns on how accuracy can be misleading with imbalanced data and that importance of TP and TN can be different, which motivates us to balance data at first and use more measures for evaluating results.

But this work is still of some drawbacks. Though they did a lot data preprocessing at first, the features are still too 'raw': they did not conduct thorough data preprocessing as we do shown in the following section. And we will see that such preprocessing is crucial in improving the performance of models. Meanwhile, they only considered one model called CatBoost. Instead, we tried out various popular methods and made comparisons .

Another experiment provides a detailed introduction to the dataset and visualization of many features [1], which helped us a lot when we were trying to understand the dataset and select features.

3 Data Set

The dataset is from Kaggle database shared by Wendy Kan[3], which contains complete (2.26 million) loan data entries for all loans issued through the 2007-2015, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. There are originally 144 features in the dataset, 39 of which have more than 80% missing values, which we plan to discard. For the rest of the features, we plan to hand pick features that are useful for our problem. We start with some features that could be highly related to loan status and picked 20 raw features to be considered in our models (see Table 1).

3.1 Response Variable

The response variable loan status is converted to a binary (1 for good loan and 0 for bad loan) variable. The reason for such conversion is that the purpose for this project is mainly to evaluate the default risk of loans, it does not really matter the type of the default.

The response variable is quite unbalanced with 563311(88.9 %) good loans and 70261(11.1%) bad loans. Given the abundance of data, we down-sampled the data by randomly drawing 70261 good loans from the original data with no replacement and then constitute a new balanced dataset of samples with 50% good loans and 50% bad loans.

3.2 Feature engineering

3.2.1 One Hot Encoding

Some features are of string type and in a small set of values. Like feature "Grade", there are only 7 possible values: "A", "B", "C", "D", "E", "F" and "G". Note that the difference of effect between two consecutive

Feature name	Feature explanation	Transform
loan_amnt	Total amount of loan taken	Log
int_rate	Loan interest rate	Log
grade	Grade of the loan	Dummy
emp_length	Duration of employment	Categorize
home_ownership	Type of ownership of house	Dummy
annual_inc	Total annual income	Log
term	The number of payments on the loan (36 or 60 months)	Dummy
last_pymnt_amnt	Last total payment amount received	Log
mo_sin_old_rev_tl_op	Months since oldest revolving account opened	-
mort_acc	Number of mortgage accounts	-
pub_rec	Number of derogatory public records	-
delinq_2yrs	The number of 30+ days past-due incidences of delinquency past 2 years	-
purpose	A category provided by the borrower for the loan request	Dummy
mths_since_last_delinq	The number of months since the borrower's last delinquency	-
inq_fi	Number of personal finance inquiries	-
open_acc	The number of open credit lines in the borrower's credit file	-
pub_rec	Number of derogatory public records	-
pub_rec_bankruptcies	Number of public record bankruptcies	-
verification_status	Indicates if income was verified by LC, not verified, or source verified	Dummy
total_bc_limit	Total bankcard high credit/credit limit	Log

Table 1: Feature selected for economic intuition

grades are not necessarily the same, it's not optimal to transform the string into integer. Instead, we can use dummy variables or one hot encoding, and to avoid perfect multicollinearity, we discard one dummy variables from regression. Pandas provide a method to directly hot encode a variable. In addition to "Grade", we also apply one hot encoding to "home_ownership", "verification_status" and "purpose". When the size of a category is too small, we merge it with another similar group. In the example of "purpose", we choose to specify three major categories: "debt_consolidation"(51%), "credit_card"(21%) and "other"(28%), because some categories contain too little data.

3.2.2 Log transformation

First we log transform all the features that specify an amount of money such as income or payment, along with interest rate. The reason is that especially for a method with linear form, a symmetric distributed feature is always desired for avoidance of heteroscedasticity. And features specifying money are very likely to be skewed towards low amount with extremely high outliers, hence a log transformation can help make the feature more symmetric. But a more general method, Box-Cox transformation, will not be exploited in this project since log transformation has been doing well and the coefficient of log transformation will be of more economic meanings.

3.2.3 Feature scaling

The last step for feature engineering is to scale the data by calling StandardScaler from sklearn.preprocessing. Some methods like logistic regression is variant to scale, while some other tree-based methods like random forest is invariant to scale, thus we choose to standardize the features at this point.

3.3 Feature analysis

In total, we have number of observations $n = 140261$ after dropping NA and balancing and number of features $p = 29$ including dummy variables. And we will add more detailed feature visualization analysis here for a closer look.

The heatmaps in Figure 1 below plots correlation between the 13 numerical features before scaling, and full 30 features after preprocessing (encoding, transformation and scaling as mentioned in previous sections). Figure 2 plots the correlation between the target feature $loan_status$ with all other features after preprocessing. We can see that grade, interest rate and last payment amount are highly correlated with loan status.

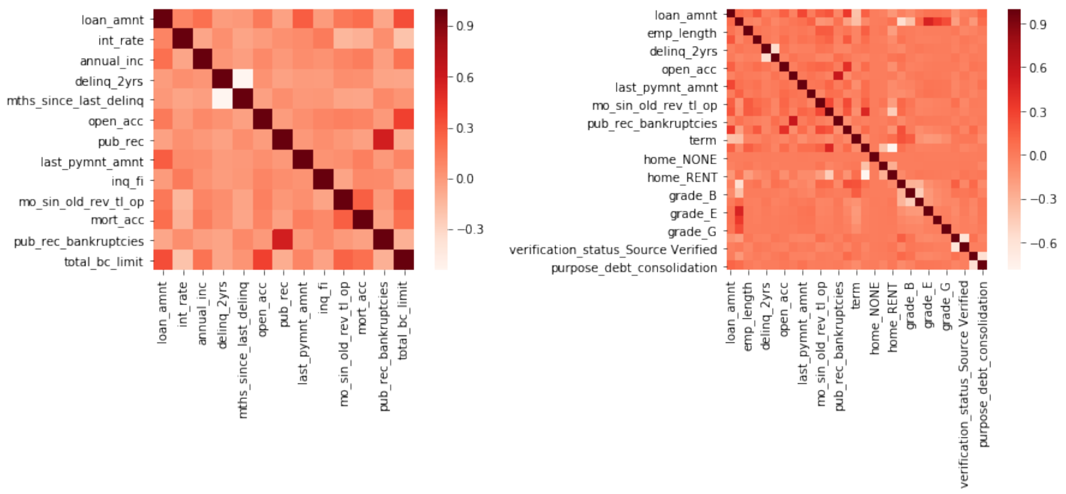


Figure 1: Correlation heatmap between features before and after preprocessing

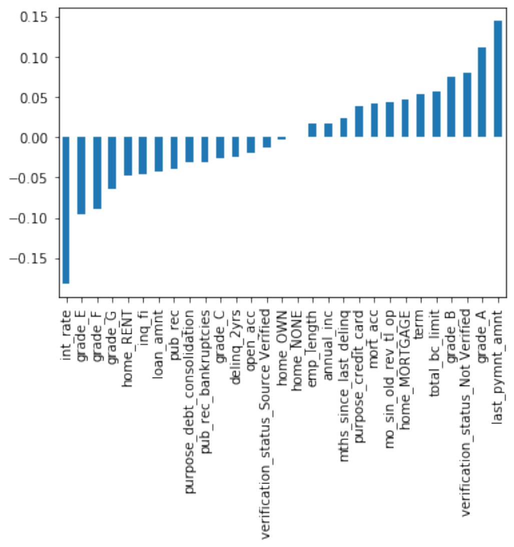


Figure 2: Correlation between target feature loan_status and other features

4 Problem Formulation

Our goal is to develop a risk model that predict good loan from bad loans based on information of the borrower. Since the amount of good loans is much greater than the amount of bad loans, i.e. the dataset is highly imbalanced, we preprocessed the data to get a balanced dataset following the steps in the previous section.

With balanced dataset, we still have to consider the importance of TN and TP in addition to overall accuracy. To address this, we can change the corresponding weights (w_0, w_1) for loss of false positive and false negative. Suppose we are forming a loss function for classification (but the specific form may vary method by method):

$$Loss = \frac{1}{n}\sum_{i=1}^{n}\left(w_0 1_{\{y_i=0\}} + w_1 1_{\{y_i=1\}}\right) 1_{\{y_i \neq \hat{y}_i\}}$$

But since we do not really know how much is TP (predict bad loan correctly) really more important than TN (predict good loan correctly) in real life, we decide to set equal weights for TP and TN when implementing models, and we leave the decision on the relative importance to banks or investors who have more economic senses. Below we present the precision, recall, F1 score and overall accuracy for comparison.

5 Methods

The first step is to shuffle the data and split them into training data and test data. And with training data and test data, we're employing different methods independently, which are logistic regression, support vector machine, AdaBoost, XGBoost random forest and neural network. For some of them, we apply grid-search on 5-fold cross-validation to tune hyperparameters. Then we compare the results of different methods and discuss the advantages and disadvantages of each method. All hyperparameter values that are not mentioned in the following context are the default values of the corresponding packages under the latest version.

5.1 Logistic Regression

Since we are calculating the probability that an applicant will be assigned to a grade, we want to use logistic regression, a simple linear model as a baseline to which we compare performance of other models with. Parameters we can tune include regularization type, strength, number of iterations and solver. We use *LogisticRegression* function from *sklearn.linear_model* package

5.2 Support Vector Machine

We will employ Support Vector Machine here as a discriminative classifier which has good performance on data that are linearly separable. Also, SVM could deal with non-linear data set by using 'kernel trick' that implicitly maps the input to a higher dimensional space. We will try kernel 'rbf', 'poly' and 'linear' in our experiments. For python implemention, we try kernel = 'rbf', 'poly' with degree 3 and 'linear' with *sklearn* pacakge and API *sklearn.svm.SVC ()*.

5.3 Adaboost

Adaboost has advantages in choosing different types of weak learners and usually yields high accuracy. Apart from that, the training process of Adaboost picks up only those features that are able to improve predictive accuracy of the model, which can help drop out large amount of irrelevant features. However, this method is regarded as a time-consuming one. In our problem, since we have $p = 29$ hand pick-up features some of which might not be closely relevant. We expect this method to have a decent performance. We write a loop w.r.t 'n_estimators' feature to pick up the best number of estimators which yields the highest accuracy for 5-fold cross validation. Then, we plot the accuracy curve for CV and test data set as a function of 'number of estimators' to observe how these two types of accuracy variate as one hyperparameter changes. Here, we use *AdaBoostClassifier* function from *sklearn.ensemble* package, *DecisionTreeClassifier* function from *sklearn.tree* package, *cross_val_score* function from *sklearn.model_selection* package.

5.4 XGboost

XGboost ('eXtreme Gradient Boosting') are fundamentally the same as Gradient boosting except for that it's usually much faster than standard gradient boosting since XGboost is quite memory-efficient and can be parallelized. So we will use XGboost method as it is superior to standard gradient boosting. For package reference, we import package *xgboost* written in Python and use API $xgboost.XGBClassifier(random\_state = 1)$.

5.5 Random Forest

Random Forest independently train each trees with a random subset of the data. Hence, this method is regarded as very robust. Since our data is very noisy, this method would produce a reliable result. Random forest has wide applications and are related to k-nearest neighbor algorithm in that both of them can be viewed as weighted neighborhoods schemes. We use *RandomForestClassifier* function from *sklearn.ensemble* package. In addition, we use *GridSearchCV* function from *sklearn.model_selection* to tune hyperparameters such as 'max_depth', 'n_estimators' based on grid search.

5.6 Neural Network

Neural Network is one of the most powerful machine learning tools that is able to deal with data of strong non-linearity. We implemented a feed-forward neural network using Pytorch. The network structure is shown in Table 2. We loaded the data using the Dataset class in Pytorch. Over each epoch, we train the model on the training data and evaluate using the validation data. We then tune hyper-parameters on the neural network using validation performance, and apply the best parameters on the testing set. Because this is a classification task, we chose the *nn.CrossEntropyLoss()* as our loss function, and the adaptive learning rate optimization *algorithm torch.optim.Adam()* as our optimization method.

6 Experiments and Results

For each method, we will try different parameters for experiments and find out the best set of parameters. Then we will aggregate the results into one table for comparison.

Layer	Parameters
Linear	size (29, 64)
ReLU	/
Linear	size (64, 16)
ReLU	/
Dropout	p=0.1
Linear	size (16, 2)
Sigmoid	/

Table 2: Neural Net Structure

6.1 Logistic Regression

For logistic regression, we experimented with different solvers and penalty setting, and they yield comparable performance. Using the default solver "lbfgs" with L2 penalty, we got 70.19% training accuracy 70/17% testing accuracy. The table below shows more results.

label	precision	recall	f1-score	support
0	0.77	0.68	0.72	15908
1	0.64	0.73	0.68	12197

Table 3: Evaluation matrix of Logistic Regression

6.2 Support Vector Machine

For SVM, we try kernel = 'rbf', 'poly' with degree 3 and 'linear' with *sklearn* pacakge and API *sklearn.svm.SVC*. And we get the experiment results as follows in table 4:

kernel	rbf	poly	linear
train accuracy	0.722	0.715	-
test accuracy	0.708	0.707	-
runtime[s]	1315.6	1744.0	∞

Table 4: Experiment results for SVM

Compared to other methods, SVM takes much more time without significant improvement in accuracy. Especially for linear kernel case, model fails to converge, i.e. the runtime explodes. If we set $max\_iter = 10000$ for linear case, the accuracy is only 0.536, which we consider failure. Hence, SVM is not a good choice when dataset is large and non-linear.

6.3 Adaboost

We use this section to illustrate how we tune hyperparameters based on cross-validation and the visualization of data. We choose 'number of estimators' = {200,300,400,500,600} as the hyperparameter to be tuned in Adaboost. For all other hyperparameters, we choose the default values in Scikit-learn. The 'number of estimators' is tuned by minimizing 5-fold cross validation errors. The cv-error and test error curve vs # of estimator is plotted in Fig.3. It turns out to be that 'number of estimators' = 600 is the best tuned parameter which yields training accuracy: 0.745, test accuracy: 0.740 and cross-validation accuracy: 0.741. At the best hyperparameter 'number of estimators' = 600, other evaluation metrics is shown in Table 5,

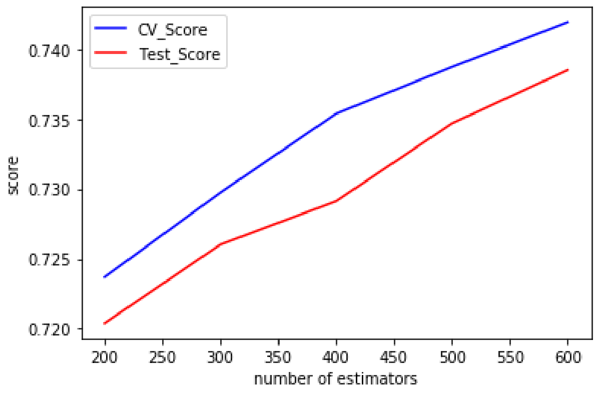


Figure 3: Score vs. number of estimator

label	precision	recall	f1-score	support
0	0.82	0.71	0.76	16351
1	0.66	0.78	0.71	11702

Table 5: Evaluation matrix of Adaboost

6.4 XGBoost

For XGBoost, we choose to tune hyperparameters 'max_depth' and 'n_estimators' using grid search method and python API *GridSearchCV* function from *sklearn.model_selection* on grid:

$$\{max\_depth : [3, 5, 10], n\_estimators : [400, 600, 800]\}$$

The optimal hyperparameter is {'max_depth': 5, 'n_estimators': 600} with test accuracy 0.766, train accuracy 0.804 and total grid search time 1737.2s. And based on the best parameter, we have the following evaluation table 6.4

label	precision	recall	f1-score	support
0	0.83	0.74	0.78	15832
1	0.70	0.80	0.75	12221

Table 6: Evaluation matrix of xgboost

6.5 Random Forest

Here, we want to illustrate how we exploit grid search to tune multiple hyperparameters. We choose the following hyperparameters used for grid search

$$\{max\_depth : [3, 5, 10], n\_estimators : [400, 600, 800]\}$$

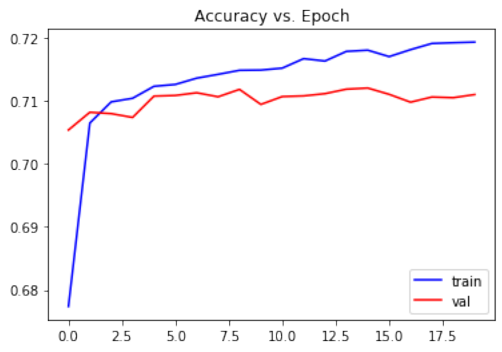
The best parameter is: {'max_depth': 10, 'n_estimators': 400} with test accuracy 0.715 and train accuracy 0.731. Based on the best parameter, we can get evaluation matrix,

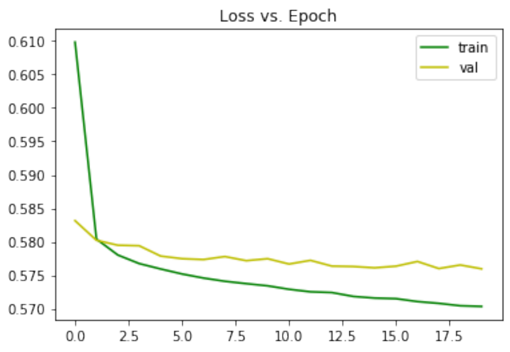
label	precision	recall	f1-score	support
0	0.83	0.68	0.74	17174
1	0.60	0.78	0.68	10879

Table 7: Evaluation Matrix of Random Forest

6.6 Neural Network

After tuning hyperparameters using validation performance, we train the neural network for 20 epochs, at batch size 96, and a learning rate of 0.0005. The plots below show accuracy and loss for training and validation data over each epoch.





After training, we apply the trained model on the testing set, and the accuracy was 71.26%. More evaluation is shown in Table 8

label	precision	recall	f1-score	support
0	0.69	0.77	0.73	14053
1	0.74	0.65	0.69	14052

Table 8: Evaluation Matrix of Neural Network

6.7 Best result for each method

In this subsection, we aggregate the best results from all methods that we implemented in table 6.7.

Method	Logistic Regression	SVM	Adaboost	XGboost	Random Forest	Neural Network
Train Accuracy	0.702	0.722	0.745	0.804	0.731	0.720
Test Accuracy	0.702	0.708	0.740	0.766	0.715	0.726
Runtime [s]	0.829	1315.6	84.8	77.5	780.0	28.7

Table 9: Best result for each method

Here, we also show the ROC curve for three models: Logistic Regression, Neural Network and XGBoost for comparison in figure 5. The ROC clearly shows a dominance of XGBoost over Logistic Regression and Neural Network no matter in terms of AUC or the shape of ROC. And since we do care more about the domain with low FP (accepting bad loans), the graph shows that XGBoost also generates a higher TP (accepting good loans).

The confusion matrices for the aforementioned methods are also plotted. The row represent actual values with the rows specifying the actual values and columsn specifying the predicted values. Here we see that XGboost produces the least number of false positives (classifying bad loans as good which we mentioned previously to especially avoid).

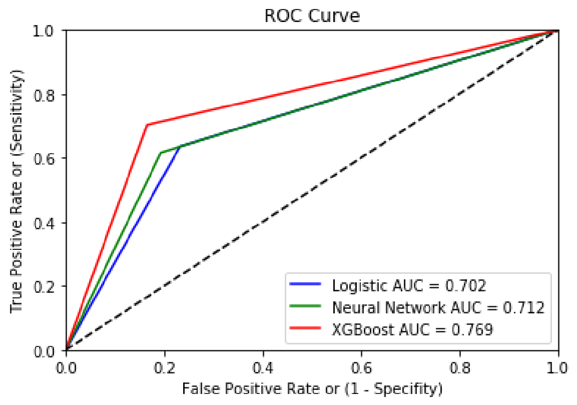


Figure 4: ROC curve for 3 methods

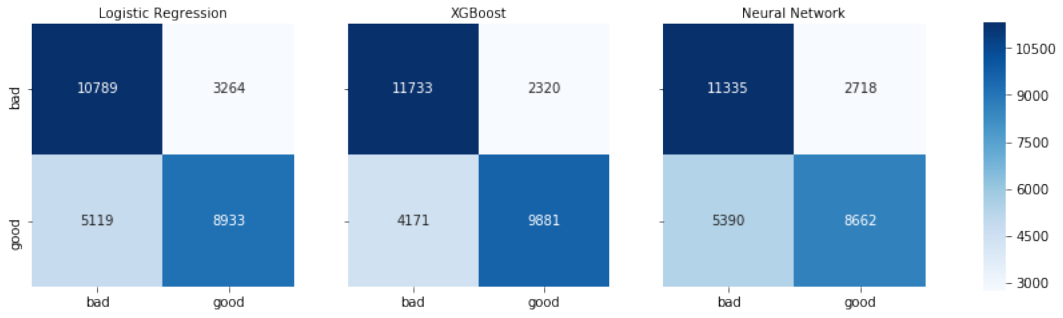


Figure 5: Confusion Matrix for 3 methods

7 Conclusion and Discussion

In this section, we will compare the results of different methods in detail and also compare with the result by other existing post in kaggle forum. From section 6.7 we know that XGboost has the best accuracy. This is probably due to its strong ability of learning complex nonlinear decision boundaries through boosting. Adaboost has the second best accuracy because it's generally used as a nonlinear classifier. On the other hand, SVM works very poorly especially under linear kernel, which also shows that our data set has strong nonlinearity w.r.t to features that should be classified by a nonlinear boundary. For the runtime of model training via each method, SVM and Random Forest costs more time than other methods (around 10^3 second) while logistic regression could finish within 1 second. Adaboost and XGboost costs similar time because they are all based on boosting methods. As a trade off between accuracy and runtime, XGboost yields the overall best performance for our problem.

For this project, we reviewed some classification methods from class as well as discovered some new methods like XGBoost. From the experiments we conducted, we gained more empirical insights in the advantage and disadvantage of different methods. When the methods we learned in class did not really stand out, we tried to explore some new methods, for instance, XGBoost performed very well in terms of its accuracy and performance. And we also learnt that not every method can work well under any circumstances, e.g. SVM did not perform well. Through this project, we also learned that in addition to the model itself, every step from feature selection, data preprocessing, hyper-parameter tuning greatly affects performance. And when we evaluate the method, accuracy is not the only metric we should be looking at.

Comparing our results with the current work found on Kaggle, our model performs better than most work we found considering that we used a balanced dataset. Despite our best effort, we achieved 76% accuracy on a binary classification task using only 29 processed features (picked from 20 raw features). Future work could exploit other features available in the dataset, or novel preprocessing method to better predict good vs. bad loans.

References

[1] Janio Martinez. Lending Club —— Risk Analysis and Metrics. `https://www.kaggle.com/janiobachmann/lending-club-risk-analysis-and-metrics`.

[2] Pavlo Fesenko. Minimizing risks for loan investments. `https://www.kaggle.com/pavlofesenko/minimizing-risks-for-loan-investments`, 2019. Online; accessed 10 November 2019.

[3] Wendy Kan. Lending Club Loan Data, Analyze Lending Club's issued loans. `https://www.kaggle.com/wendykan/lending-club-loan-data`, 2019. Online; accessed 10 November 2019.