Lending Club Borrower Behavior Prediction by Using the Techniques of Machine Learning

Sponsored by Connexin Group LLC

GROUP MEMBER


YUFEI GAO


ZHEJIAN PENG


XIANZHEXU


YULONG LIU


ZIXIAO WANG


YANXIN LI

Table of Contents

Preface

Predicting borrowers' potential behavior is always a challenging topic for each lending-related companies or platforms and our main task of this project is conducting a set of exploratory analysis and trying to apply various machine learning techniques to predict possible lending outcomes of default or fully-paid based on the utilization of data provided by Lending Club.

Before we get started with the details of our model, let us illustrate a simple example first.

Borrower A:

loan_amnt	funded_amnt	grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d
3250	3250	D		< 1 YEAR	RENT	6000	NOT VERIFIED	Nov-2009

Borrower B:

loan_amnt	funded_amnt	grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d
20250	20250	C	Archdiocese of G-Houston	3 YEARS	RENT	43370	VERIFIED	Dec-2011

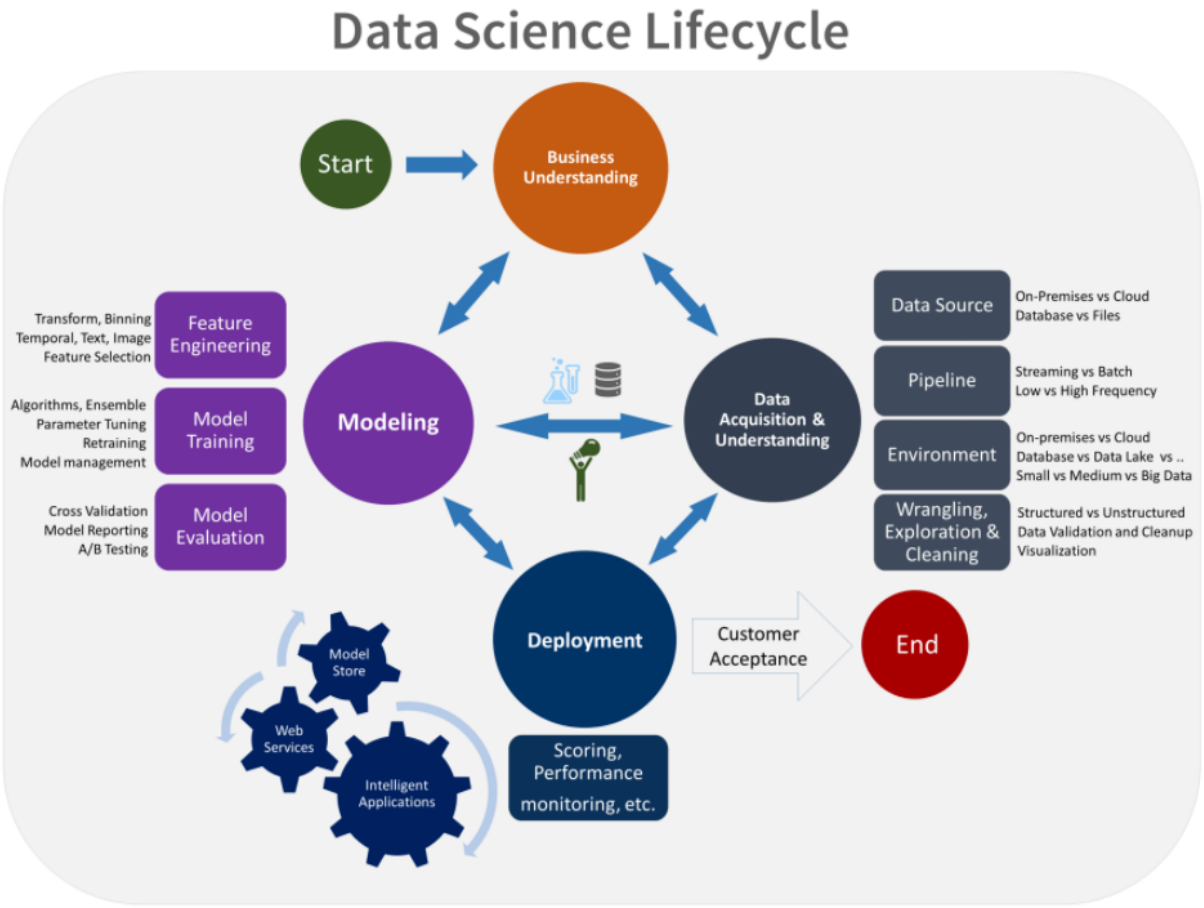
The two tables above reveal the personal information of two individuals, A and B. At first glance, their information share some similarities, such as they both trying to apply the amount of money which is about half of their annual income, they both currently renting a living place and both of them have low loan grade estimated by Lending club. Also, their information still has some difference, for instance, borrower B has a longer length of employment than A and his/her source of income had been verified by LC. These two persons are exactly two cases I draw from our dataset but I only present partial information right here. As result, the final status of loan lent to A is charged-off but borrower B fully paid his/her loan even these two borrowers have similar information for the loan application.

In reality, our data structure is much more complicated than this one. Our dataset has over 150 features for over 1.8 million applicants and some features are significantly missing, some of the features have no meaningfulness to utilize on our model and some features cannot be taken simultaneously which under the consideration of information leakage. Hence, it is worth us taking certain amount time to interpret and evaluate the quality of each feature rather than simply call the programming package do this for us.

Keep working on the iterating process of data cleaning and optimization, we attempt to construct this prediction model by taking methodologies of logistic regression test and deep learning. In the following sections, we will go through the details of each stage of our work and show how we gradually build a robust prediction model while keep avoiding common problems of data exploration.

Introduction

Our ultimate goal of this project is trying to build a robust model for predicting whether the borrowers of the loan will repay the loan completely or would default based on features about his/her financial status. We mainly followed the data science lifecycle illustrated below and keep working on the iterating process of model optimization and evaluation.



Starting with features understanding part, the goals of this stage is trying to specify the key variables that are to serve as the model targets and whose related metrics are used determine the success of the project. In order to handle this large size of date in more careful manner, we keep working on the process of features classification, anomaly detection and model selection in the first couple of weeks. Deeply understanding the data information is always an important object for us to produce a clean, high-quality dataset whose relationship to the target variables is understood, and this is the main content of the section of data acquisition.

There are several important steps of part of data acquisition, such as:

- Ingest the data: Set up the process to move the data from the source locations to the target locations where we can run the analytics operations, like training and prediction
- Explore the data: Use data summarization and visualization to audit the quality of our data and provide the information we need to process the data before it is ready for modeling. At this point, we implement the technique of Tableau visualization which has significantly facilitated our process of data interpretation and relationship figuration. In the meantime, we constantly investigate our data for the sake of preventing the problem of information leakage, which is requiring iterating process between building an analysis data set, creating a model, and evaluating the accuracy of the result.

- Set up a data pipeline: Conduct a process to score new data or refresh the existing data regularly as part of an ongoing learning process. The types of the pipeline we used in our model are Batch-based and hybrid as mentioned in previous sections.

The stage of Modeling focuses on determining the optimal data features for our machine-learning model and makes our model as informative as possible on its prediction capability. And this is realized by our rigorously training on our model, which involves model selection, splitting and converting original data, evaluating training and test set and determine the best solution based on our model validation.

The last step is model deployment which is about deploying our models with a data pipeline to a production-like environment for final use acceptance. We strive to construct a model that our users can simply input the information for each final feature we preserved and get the theoretical probability of borrowers' default outcomes. And we visualized this process on the dynamic dashboard of Tableau for the purpose of this section.

Data Source and Understanding

The data used for this project range from 2007 to 2017. The dataset comes with a total of 891283 observations and 144 features. After data processing and feature engineering, 851641 observations and 57 features are used in our modeling process.

In order to select features used for modeling, we examine the economic meaning of each feature and proceed with graphical analysis to see which features are important in explaining default. In the process, we are also able to identify features which contain information that is not available at the issuance of the loan, and hence should not be used for predicting default.

Important features

1. Loan term
   From 2007 to 2009 only loans with a term of 36 months are issued. Since 2010 loans with a term of 60 months are issued, on average these loans account for 25.51% of loans issued each year.

   Overall the percentage of loans defaulted and charged off with a term of 36 months is lower than that of 60-month loans, with values of 16.01% and 31.08% respectively. Looking on a yearly basis, other than 2007, 2008, 2015 and 2016, the percentage of defaulted and charged-off loans for 36-month-loan is at a level of around 12.47%, excluding data in 2017 where the number of defaults is understated as the majority of loans issued in 2017 are not due yet.

   For 60-month loans, it's more meaningful to look at 2010, 2011, and 2012, as loans issued in 2013 or later are not due yet and the percentage of default is thus a priori which is understated. In the year 2012, even though 60-month loans account for as low as 18.54% of loans issued in the year and default rate for 36-month-loan is around the average level, the overall default rate in the year is increased.

2. Purpose of the loan
   Out of the ten-year records, 59.09% of loans are for debt consolidation, followed by 21.36% of loans for debt financing. The proportion of loans is relatively stable over the years, with loans for debt consolidation making up the majority of the loans issued each year.

Overall, loans issued for funding small business, renewable energy, moving, medical purpose, and educational purpose have the highest default rate, however, the impact is immaterial as these loans account for less than 3% of total loans issued. Loans for debt consolidation, which constitutes more than half of total loans issued, have similar default rate which is as high as 20.76%.

3. DTI ratio

DTI is calculated as the borrower's total monthly debt payments on the total debt obligations (excluding mortgage and the requested lending club loan), divided by the borrower's self-reported monthly income. It indicates the solvency of the borrower.

It can be seen from the graph that there are many outliers for DTI in each category and this might be due to the fact that monthly income is self-reported. Nonetheless, the trend that fully paid loans on average have lower DTI can be observed.

4. Length of employment

Majority borrowers have over 10 years of employment history. The rate of default is lowest for this group. In contrast, borrowers who are unable to provide employment history has the highest default rate.

5. Annual income

It is manifested that the higher the income, the lower rate of default.

6. Number of delinquencies in last two years

The number of delinquencies in last two years denotes the number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years. It is also an indication of financial health. The higher the number of delinquencies in two years, the higher the rate of default would be.

7. Number of inquiries in last 6 months

The number of inquiries in last 6 months illustrates how actively the borrower is in seeking financing, as the number will increase when the borrower's credit history is pulled (which constitutes a so-called hard inquiry). A higher number of inquiries is associated with greater rate of default as the borrower could be in financial distress when he needs multiple sources of financing.

8. Homeownership

It can be seen from the graph that borrowers who rent have the highest probability default, followed by borrowers who specify others, borrowers who own a place, borrowers who are under the mortgage.

9. Joint application

When there's a second applicant, the default rate is actually lowered.

Information Leakage

1. Verification

There is robust screening model which all applications need to go through. When an initial loan application passes that process, it is generally deemed as less risky.

A loan application that triggers the need for income verification could mean that the applicant in question may have an elevated risk profile. Loans facilitated by Lending Club without income verification have historically lower charge-off rates than those that were income verified or income source verified.

2. Hardship flag

Hardship is defined as experiencing financial hardship due to an unexpected life event such as a medical emergency, temporary job loss, unexpected car or home repairs, death in the family, or other events (natural disasters).

It is subject to the approval of Lending Club for borrowers to qualify for hardship plan to temporarily make interest-only payments to accommodate. According to Lending Club, borrower's application for enrollment in hardship plan is likely to be approved when Lending Club think the borrower is likely to make repayments. As such, for loans with hardship flag, the default rate is as low as zero.

3. Outstanding principle

Upon examination, for all fully paid loans, the outstanding principal is 0. Loans with unpaid balances are deemed as default.

Data processing

We separately download the dataset from the lending club website. Because of its huge size, we can hardly use excel to combine them. So, we use pandas to read data 100000 rows once and then combine them together. Firstly, we decide to set "loan status" as "y" variable, namely the dependent variable and the other features as "x" variable because our purpose is to predict whether the loan will be defaulted or not. In terms of "y" variable, we drop all data, whose loan status is current, because their final statuses are not clear now so that they cannot be trained or tested. Additionally, we drop data whose loan statuses are "grace period" and "late" because their accumulative proportion is less than 10 percent, which may bring about chaos if we put them in trains set. Furthermore, we combine the “default” and “charged off” into “default”, “fully paid” and “fully paid: does not meet the policy” into “fully paid”. So, we get a “y” variable with binary class.

Then we will deal with multiple “x” variables, namely the independent variables. Firstly, we check each economic meaning of more than 150 features totally and then drop those features, which are obviously uncorrelated with loan status (Eg. Member ID, URL) and those with evident information leakage, namely the information of those features we cannot get when predicting the loan status in the future. (Eg. The interest rate, the term of the loan, and installment) We also use ‘sys’ package in python to draw the correlation matrix of the features and we do not find apparent pairs of features with high correlations.

After that, we reasonably convert some non-numerical features into numerical features because we can hardly train and test dataset with non-numerical features with most machine learning models. For categorical features, we use python’s function “().astype" to deal with them. For example, one feature is called ‘application type' and its values are whether "joint" or "individual". Then Python will create two new binary features, which are called ‘application type: join' and ‘application type: individual' and drop ‘application type' as well. Afterward, we think that zip code cannot fully explain the pecuniary condition of each person, so we find the mean and variance of income information of each zip code and join these two new features into the dataset.

At this point, we have roughly constructed our dependent features and independent features. However, we still have to deal with missing data and outliers. Firstly, we drop features, whose missing percent are more than 20 percent. Then we check outliers of all initial numerical features, which may cause problems in the normalizing step afterward. For those ones with distinct outliers, we respectively set a threshold for them and drop the data, which exceed the threshold. After that, we normalize all initial numerical features in order to more efficiently train and test

them. Specifically speaking, we normalize the test set with mean and variance of train set instead of its own mean and variance to avoid information leakage because we regard that we should not know any information of test in advance. Finally, we fill the remaining missing data with zero, which equals to fill the blank with mean before normalizing.

Finally, we randomly split 75 percent data as train set and remaining 25 percent data as the test set. In short, we have completely constructed our train set and test set for machine learning models.

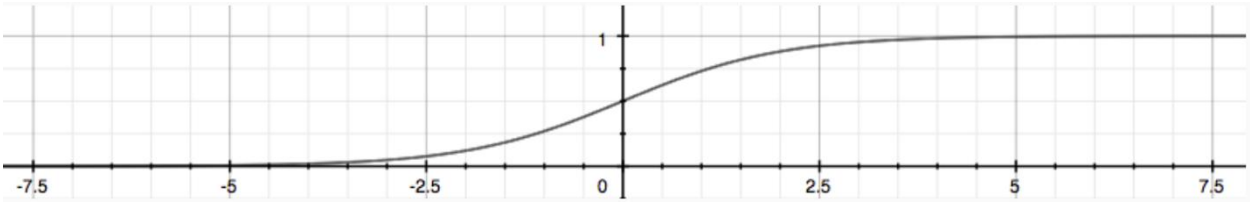
Logistic regression

Introduction to Logistic regression and Lasso regularization

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. In our project, since our output only consists of fully paid and charge off, we will focus on the binary classification problem in which y can take on only two values, 0 and 1.[1]

Intuitively, it doesn't make sense for $h_\theta(x)$, the predicted output, to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$. To fix this, we change the form for our hypotheses $h_\theta(x)$ to satisfy $0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging $\theta^T X$ into the Logistic Function, also called "Sigmoid function". The equation and the picture of sigmoid function are shown below. [2]

$$g(z) = \frac{1}{1 + e^{-z}}$$

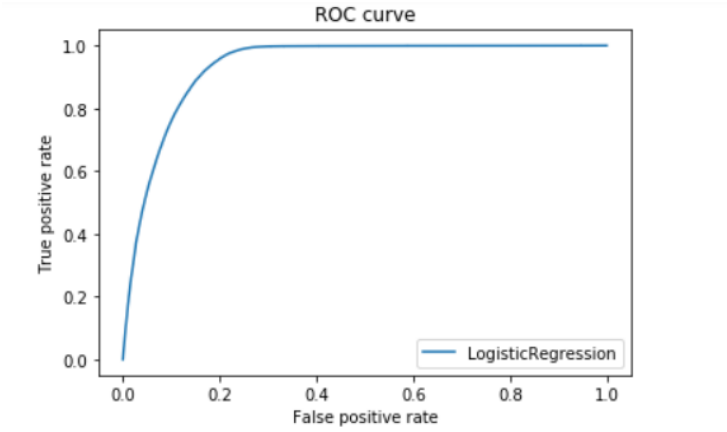


In order to prevent multicollinearity, we add a penalty term to our Logistic regression. Just like Lasso regularization, which add l_1 norm ofβ, the regression coefficients, to linear regression, we also add l_1 norm of β to our Logistic regression.

Regression result

We train and fit the train set using Logistic regression, and then we use the model to predict the test set. We use several indicators to show the prediction results. First, in order to simplify the explanation, we denote True positives as TP, True negative as TN, False positives as FP, False Negative as FN. The first indicator is confusion matrix, which shows the actual result and probability of TP, TN, FP, and FN. Then we have mean accuracy = (TP+TN)/Total, specificity= TNR=TN / (FP + TN) = 1 – FPR, sensitivity (TPR)=TP / (TP + FN), FPR= FP / (FP + TN), and G-score = sqrt(sensitivity*specificity). Finally, we use ROC curve, which shows the relationship between false positive rate and True positive rate, to show the result. The model could have a better prediction result if the ROC curve is closer to the left top corner.

The following picture is our model result.




```
Area Under the Curve (AUC) from prediction score is 0.934670
G score is 0.873911
Specificity is 0.788450
Mean accuracy score is 0.826984
Confusion Marix
[[101930  27349]
 [  1103  34065]]
```

In order to prevent information leakage problem, we drop the variable out_principle, which is the remaining principle have not been paid. Because we use historical data, if the out_principle is none zero, the loan status is charged off. As a result, the out_principle variable is highly correlated to loan status and should be dropped.

Model Validation

After building this model, one step that we need to take is to validate its correctness. We check the model from its assumptions and data quality, then we will run the model with new input dataset.

Here are some assumptions for the Logistic regression model. Firstly, since this is a binary logistic regression model, it requires the dependent variable to be binary. We have already set 0 and 1 for the two categories "default" and "fully paid" respectively. Then, the model requires that observations should be independent of each other, the observations should not come from repeated measurements or matched data, which means two loans should not under the same member id in this problem. Since we cannot find any information about member id from the website of Lending Club, we have to assume that they are not related to each other. For the third assumption, they require that there be little or no multicollinearity among the independent variables. We have built the covariance matrix to calculate their covariance, thus to make sure any independent variables left have low correlation with each other. Finally, they require that the dataset be large enough. We have already found all the data we can use from the Lending Club, and it contains over 800,000 observations, we believe this dataset is large enough to train the model.

Next, we will focus on the data quality. We believe that after the data processing part, the dataset we determined to remain is with quite high data quality. During the data processing part, we have already excluded features that we should not use as independent variables, which are features that will lead to information leakage. Since Logistic regression requires all its input variables to be numerical types, we transfer date of other types, (like string type), to numerical type. Also, we deleted features with huge amount missing data, for example, the "hardship" categories, and filled in the missing data in the rest features. Finally, we calculated the number of outliers for each feature by using the Inter-quartile Range method. Then deleted features with a huge amount of outliers.

We tested the model with new input dataset. Firstly, we created two very simple datasets. Denote the independent variable as X, the dependent variable as Y. The first dataset is: all features in X are value 0, with value 1 in Y; all features in X are value 1, with value 0 in Y, both classes contain 100,000 samples. The second dataset: all features in X are value 0, with value 0 in Y; all features in X are value 1, with value 1 in Y, both classes contain 100,000 samples. We expect the

model will give us 100% accuracy since in each case, classes are very easy to separate. And the result really meets our expectation, the accuracy for both classes in both cases are 100%.

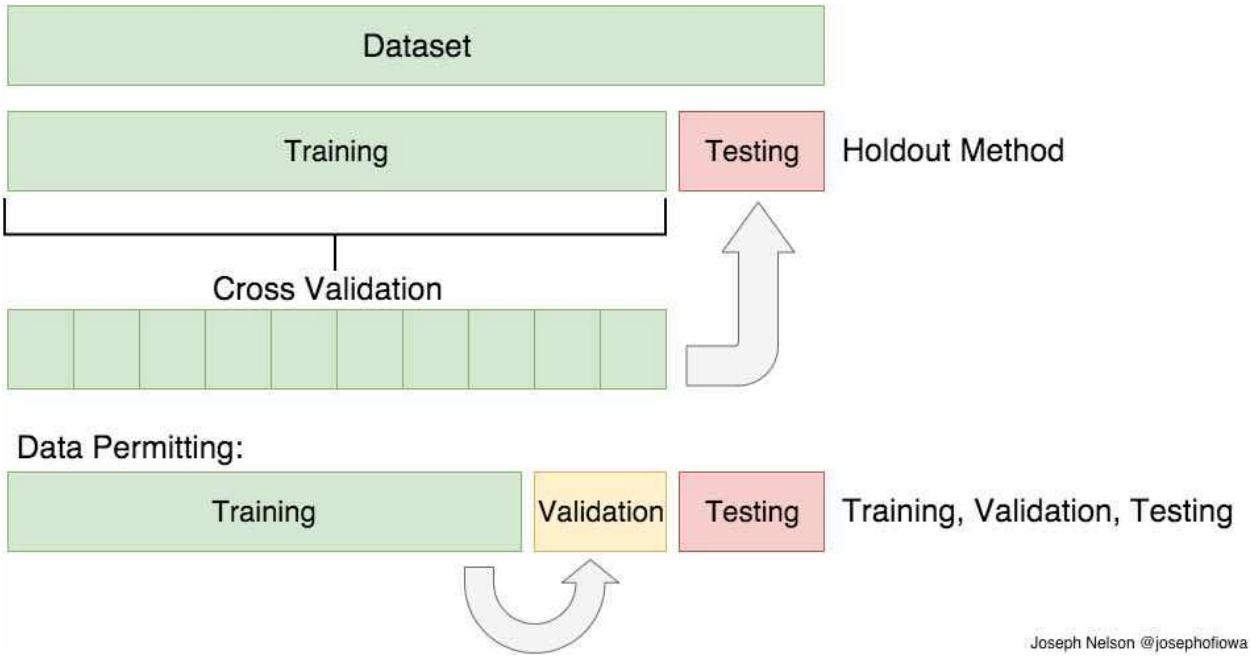
Next, we use the dataset from the Kaggle website. It contains different features and observations from what we used. We just did some necessary data processing, for example, deleted features that will lead to information leakage, transferred the data type to be numeric, and filled in features with missing data. Then we use this dataset to run the model. The result is also quite well, with AUC, G score and mean accuracy is around 0.95, 0.89, and 85.8% respectively.

To sum it up, we have reviewed the data quality in detail to confirm that we have proper data usage. We reviewed some key assumptions of our model, assessed the applicability of our model, so made it conceptually sound. We tried new dataset as the input thus we eliminated the possible uncertainty of the result brought about different input variables. Finally, we came to the conclusion that the model is quite robust and can perform well in this real problem.

Parameter Tuning

Another important part of the Logistic regression is the parameter tuning. In previous trials, we just use the default value for some parameters. We realized that these values may not be the "best" for our model in this specific case. Thus, we need to tune these parameters. The procedure for this part is quite straightforward, that is, we will try different possible values in a list, then train the model with these parameters, choose the one with the best performance. The key point is how can we evaluate the performance of the model with different values of parameters. We will use the cross-validation method in our model.

For general cross-validation, we further split the training dataset into the real "training" dataset and "validation" dataset, which will function as the real "testing" set. In our model, the validation dataset will be 20% of the original training set. Once we pick up a value from the list, we will firstly use the validation set as the test set to get evaluation and scores from the model. After looping over all possible values, we can get a series of scores from the validation set. Then we pick up the value with highest scores. In this model, we use the mean accuracy score as our measurement. The whole process for the cross-validation can be illustrated as below:



An advanced improvement for the cross-validation method is the K-fold validation method. The key difference between it and the general cross-validation method is that K-fold validation cuts the original training set as K part(with an equal amount of data), they run the model for K times, each time we use a different set as the validation set and the rest as the training set. We average the score of these K tests and then use it as the evaluation score. In this model, we chose K = 5. The figure below illustrates the whole process:

The first parameter we will need to tune is the parameter for regularization strength "C". In the Logistic regression, the cost function will be:

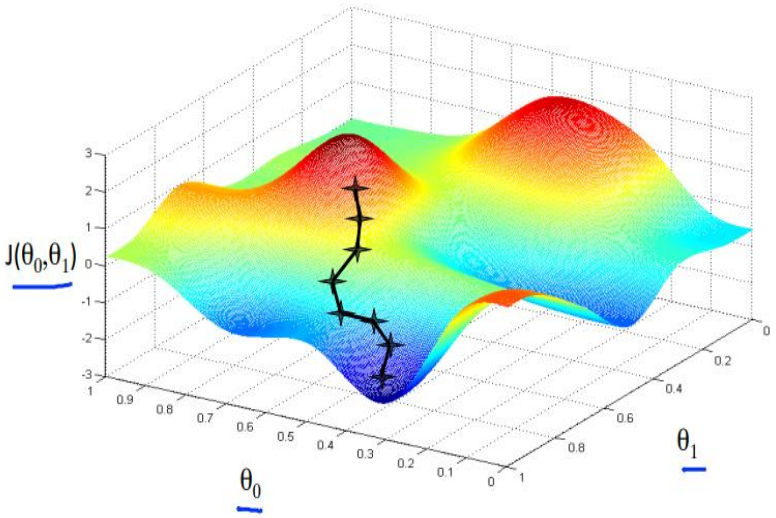
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right] + \lambda\sum_{j=1}^{n}\theta_j^2$$

C equals to 1/λ, the default value for C is 1.0. The larger C is, the less the regularization strength is. If C is too large or too low, it will lead to low accuracy. We set the test list as [0.01, 0.1, 0.5, 1, 2, 5, 10, 20], the result is shown below:

Value of C	0.01	0.1	0.5	1	2	5	10	20
Avg. accuracy%	83.263	83.275	83.273	83.273	83.274	83.274	83.273	83.274

We can see that the average accuracy has a relatively low sensitivity to C in the range of 0.1 to 20. In this model, we can set C = 0.1 according to the list above.

The second parameter we will need to tune is the parameter for the number of maximum iterations. When training the model, you will need to take step by step to find the optimal point from somewhere in the grid. Generally, you will not find the exact optimal point but instead stop at somewhere very close to the optimal point. The whole process will be illustrated below:

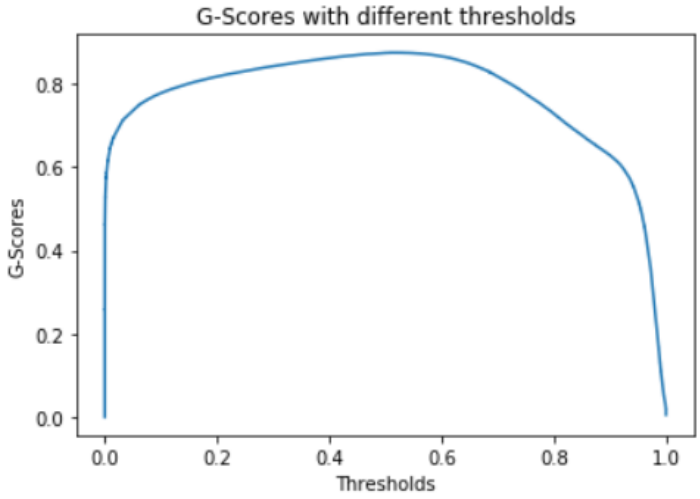


In order to stop the algorithm, we can set the number of maximum iterations, and the algorithm will stop after those numbers of steps. The default value is 100 steps. We tried 100, 200, and 500 in the following tests:

Value of Max iteration	100	200	500
Avg. accuracy%	83.33	83.55	83.84

We can find that more iteration brings about higher accuracy. So if it is possible, we should use more iteration steps as we can.

The third parameters we can tune is the threshold for the Logistic Regression. This parameter is optional to tune and the default value is 0.5. Generally, after we trained the model, we get the probability of one observation to be classified as class one. If it is larger than the threshold, it will be classified as class one, otherwise, it will be classified as class zero. We can get some G-scores with different thresholds as below:




```
The highest G score is 0.875345 with threshold at 0.511483
```

The optimal threshold in this problem is around 0.511. And we after setting the new threshold, the value will be:

Set C = 0.1	
Threshold = 0.5	83.27
Threshold = 0.5114	83.77

Set C =1	
Threshold = 0.5	83.27
Threshold = 0.5114	83.77

So we can find that the score improved a little bit under this new threshold value.

Feature Engineering

Introduction: The process of creating data features from our existing dataset by inclusion, aggregation, and transformation of raw variables to facilitate model training.

Process

- Interpreting and testing features
- Deciding what features to create
- Check how the features work with our model and improve them if needed
- Go back to the first step to creating more features until work is done

Implementation

1. $\frac{tot\_coll\_amt}{tot\_cur\_bal}$: Total collection amount ever owed / Total current balance of all account
   This feature aims to estimate the repaying ability of the borrowers and the value of this ratio is lower, it means the financial status of this particular borrower is worse and he/she is more likely to be a risky borrower.

2. $\frac{num\_rev\_tl\_bal\_pos}{num\_rev\_accts}$: Number of revolving trades with positive balance / total number of revolving accounts
   A revolving account is an account created by a lender to represent debts where the outstanding balance does not have to be paid in full every month by the borrower to the lender. You have the option to "revolve" some or all the balance to the following month and the lenders charge you interest on the amount you revolve and this is how they make money.
   Hence, the value of this ratio is higher means the certain borrower has a higher level of risk on their potential loan payment, since it could possibly be the case that the person was trying to set up a bunch of accounts and mutually pay the minimum required payment due to the condition of the different payment cycle.

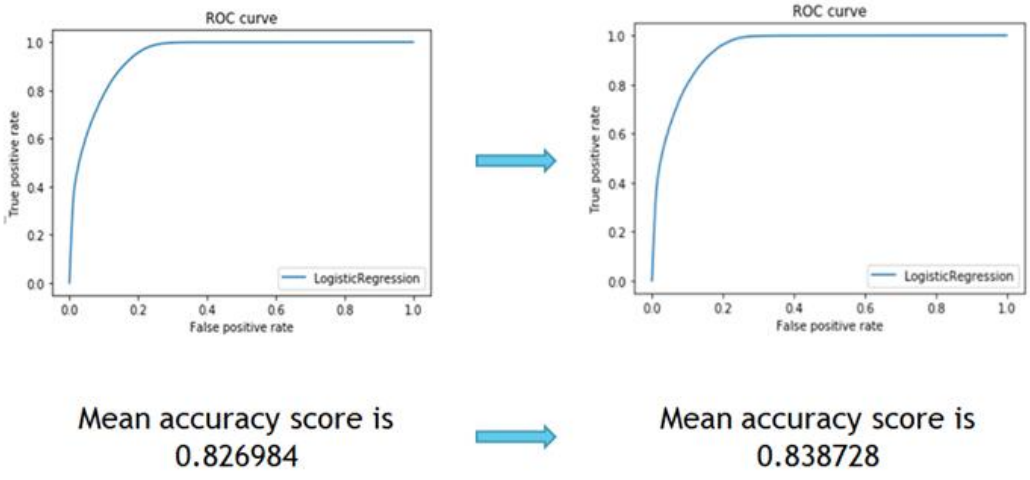
3. Classifying different month of issue date to quarter format
   This is the attempt on converting different month into four different quarters of a year. We are trying to set up a new feature which is related to external economic factors and see if it is possible to improve the accuracy rate of our model. For instance, there are

several dates always come with the shopping discount, such as Black Friday and the Cyber Monday after it. People tend to shop around the mall or outlets in their place and this would significantly motivate a certain group of people who lack money to temporally apply for the loan for this purpose.

4. $\frac{loan\_mat}{avg\_inc\_zip}$: Loan amount / average income based on zip code of borrower
   At this point, we tend to estimate the personal income level based his/her information of zip code, since the same annual income of two borrowers who living in different state or city is a totally different story behind.
   After we finished this averaging manner, we divide the averaged income by the loan amount applied by borrowers. If the value of the ratio is higher, then it would be a riskier behavior on lending money to this person based on this particular aspect.

Improved result

We added four features above into our model once we ensure the data quality and run the regression logistic test.



Deep Learning Model

Feature Optimization

Income normalization by zip code:

We want the income feature to truthfully reflect the level of income of each applicant. This means that an 80K income in Champaign is not the same as 80K income in Los Angeles, because average income in LA is much higher than Champaign. Thus, raw income data are not a good indicator of applicants' income level. Therefore, we normalized income distribution to N(0,1) distribution by each applicants' zip code. Therefore, income information reflects the income level in each applicant's living region.

Standardization

Normalize Train/Test Set Separately:

To ensure our model best reflect the real-world problem, we have separated the train/test set before data standardization. Following our assumption in data leakage check, the test set should not have any knowledge about the distribution of the data, we conduct train/test split before data standardization. We compute the mean and standard deviation of each feature and stored them in a dictionary. Then, we used the train sets' mean and standard deviation to normalize test set. In practices, when new data arrives for prediction, its features will be normalized using passed mean and standard deviation we learned from pass data.

Model Structure

Deep learning model consists of 7 dense layers with 6 dropout layers in between. The first layers consist of 57 neurons corresponded to 57 input features, 3 layers consist of 64 neurons, then 2 layers consist of 32 neurons, and the last layers have 16 neurons. There are 6 dropout layers between each layer. The output layers consist of 2 neurons for two classes of outputs. We used Adaptive Moment (ADAM) optimizer and sparse categorical cross-entropy for optimization method and the loss function.

We compare 3 model designs: gradient descent, Adam optimizer, and Adam optimizer with dropout layers. Experiments show that Adam optimizer with dropout layers shows the best performance.

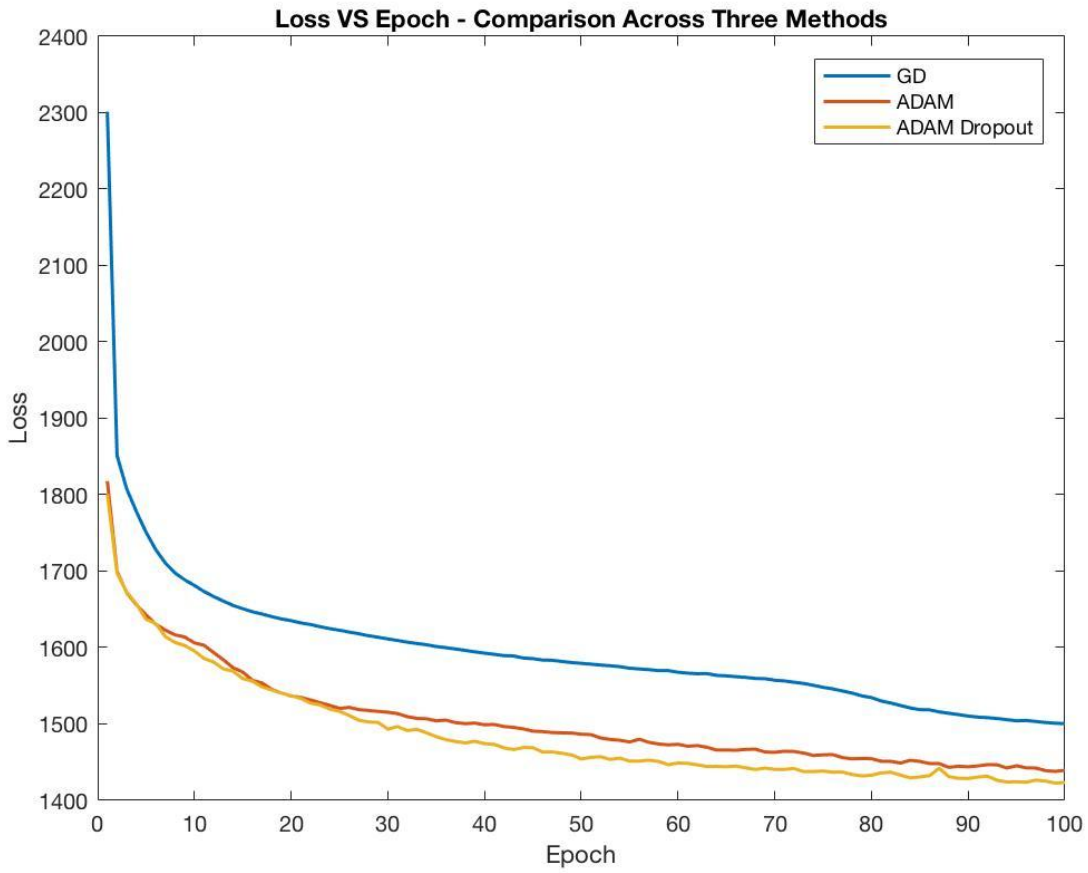


Figure 1: The ADAM Optimizer with dropout layers shows the best performance. Please note that his plot is generated using model under the TensorFlow framework. Later, we wrote and tested my final result in Keras. The loss in Keras is not comparable with this plot. This plot helped us determined optimizers and set a foundation for our final model.

Optimizer

In our deep learning model, the model is trained to minimize the **objective function** – Sparse Categorical Cross-entropy. Thus, we need to choose the best optimization algorithm for our problems. The two candidates are Gradient Descent and ADAM.

Gradient Descent: Gradient Descent (GD) tries to minimize the function f by iterative moving in the direction of steepest descent. In deep learning model, the descent is used to update parameters for neurons.

$$x' = x - \epsilon \nabla x \, f(x)$$

x is the parameter that needs update and ϵ is the learning rate.

(Goodfellow, Ian; Yoshua Bengio; Aaron Courville n.d., 84)

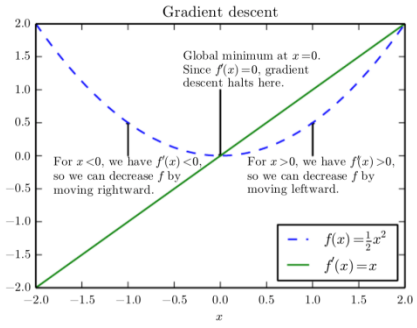
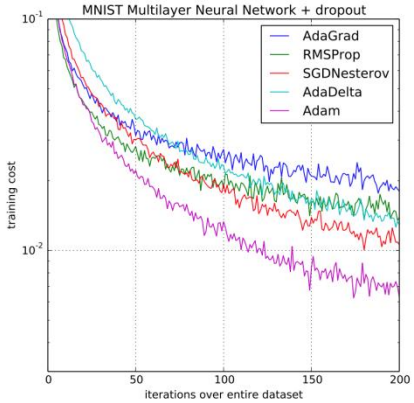


Figure 4.1: An illustration of how the derivatives of a function can be used to follow the function downhill to a minimum. This technique is called *gradient descent.*

ADAM: ADAM is a stochastic gradient descent optimizer (SGD). Similar to GD, ADAM introduced an improvement over traditional GD with fixed learning rate. Traditional GD used a fixed learning rate, while ADAM uses an adaptive learning rate. ADAM optimizer will adjust the value of ϵ as it gets closer to the minima. Thus, it generates a better result than fixed learning rate GD optimizer. ADAM also wins against other SGDs.

(Diederik P. Kingma; Jimmy Lei Ba, 2017)



Hyper-Parameter Tuning

We want to choose the best hyperparameter for our model. We first test 3 optimization method: gradient descent, ADAM, and ADAM with dropout. As we can see that the ADAM with dropout layers tend to perform the best. Also noted that this model is not our final model, we plotted this in the TensorFlow framework. Later, we wrote my final model in Keras. Therefore, the numerical value of losses is not comparable. The key is ADAM shows better performance than regular gradient descent because ADAM adjusts learning rate as the model trains, thus the model can reach closer to the optimal weight than gradient descent optimization method.

After we determine the optimizer, we used a grid search to find the best hyper-parameters on following choices: batch size: [8, 32, 64], learning rate: [0.00001, 0.0001,0.001], neurons initialization method: [uniform distribution, normal distribution], and Dropout rate: [0.0, 0.2, 0.4, 0.6]. The complete grid search result will be attached in appendix.

Limited by our computation resources, we were unable to use a bigger grid. GPU implementation might help accelerate this process. we used Amazon AWS cloud computing services to run this grid. In order to reduce the number of combinations and accelerate the process, we fixed the number of epochs to 10 and used 3-fold cross-validation. we also fixed dropout rate to 0.0 in the grid and tuned it manually after other optimal hyper-parameters have been found. My grid search returns the best hyper-parameters:

Best: 0.867080 using {'batch_size': 8, 'dropout_rate': 0.0, 'epochs': 10, 'init_mode': 'normal', 'learn_rate': 0.001}

However, we realize that this result is not optimal under our fix 10 epochs grid because the grid returned the largest learning rate and smallest batch size. This essentially means that the optimal set of hyper-parameters is the one that trains the model with most iterations and larger learning rate. However, this doesn't affect the optimal performance when you increase the number of echoes. We want a set of hyper-parameters that will return the best performance when the model slowly converges to the global minima.

Instead, we used the second-best set of hyper-parameters as the starting points.

0.867049 (0.002364) with: {'batch_size': 64, 'dropout_rate': 0.0, 'epochs': 10, 'init_mode': 'uniform', 'learn_rate': 0.0001}

Then, I increase the number of epochs to 1000 and test a set of dropout rates [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]. The test shows the optimal dropout rate here is 0.1.

Hyper-Parameters
Batch Size: batch size refers to the number of training examples utilized in one iteration. Here we used 64 as batch size, so we utilized 64 training examples in one training iteration. Usually, larger batch size provides a more accurate estimate of the gradient. **Dropout Rate:** dropout is a common way to prevent overfitting in deep learning model. Dropout layers randomly discard units in each hidden layer. The dropout rate is the percentage of neurons to be discarded in each hidden layer. We found that dropout rate of 0.1 works best in our model. **Epochs:** one epoch is the single pass of the full training set. For a stable model, a larger number of epochs will not hurt the performance.

Model Result and Validation

Our model trained using tuned hyper-parameters on 1000 epochs. It results 89.41% mean accuracy on the test set. The following two figures summarized our result on the test set. Our model result 90.14% accuracy on the train set. Our model shows little overfitting issue due to our dropout regularization.

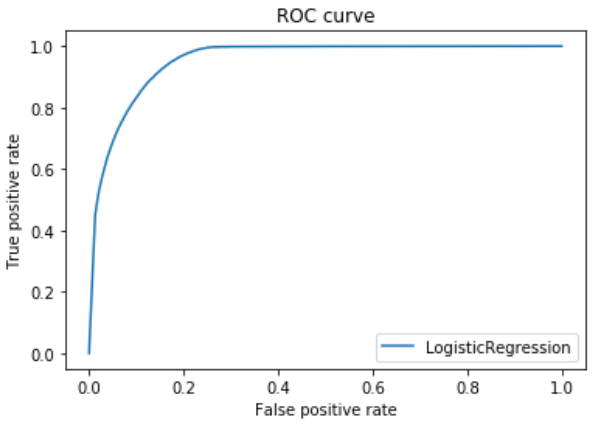


Figure 2: ROC curve on the test set. AUC = 0.953250. The x-axis in false positive rate and the y-axis is the true positive rate. There is a trade-off between false negative and false positive.

```
G score is 0.837982
Specificity is 0.930531
Mean accuracy score is 0.894144
Confusion Marix
[[157135  11731]
 [ 10807  33238]]
```

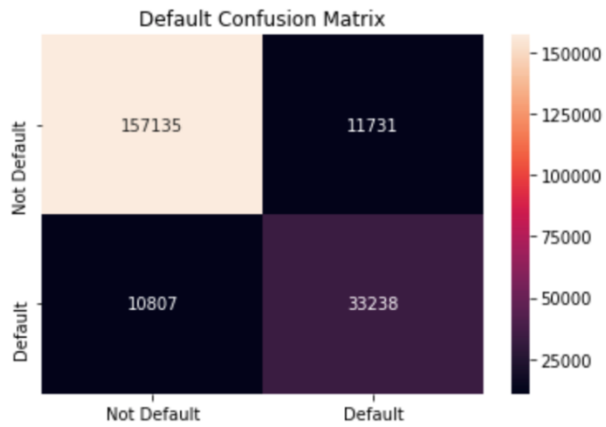


Figure 3: Confusion Matrix on the Test set

Understand Model Performance
Specificity: specificity is the true negative. It measures the percentage of not default loan who are correctly identified as not default rate. **Sensitivity:** sensitivity is the true positive rate. It measures the proportion of default that is correctly identified as default. **Accuracy:** accuracy is the percentage of correct predictions over the size of the test set

Validation

For model validation, we first test if my model can overfit the training set with enough epochs of training. The assumption is that any appropriate machine learning model can overfit the training data; otherwise, your model choice is not good for this type of problem. We validate my model by randomly choice 10% data from my total training set and perform 1000 epochs to see if my model can outfit training set to 99% accuracy. We only choose 10% of total training set because it can accelerate the process of choosing the best model with enough layers and neurons. The initial 2-layer neural network had difficult fit the training set to 99% accuracy after we increased the number of neurons and layers, my model can easily fit training set to 99% accuracy with roughly 1000 epochs of training.

We also test my model against overfitting by comparing the model's accuracy on both training set and test set, which following the model performance on the training set.

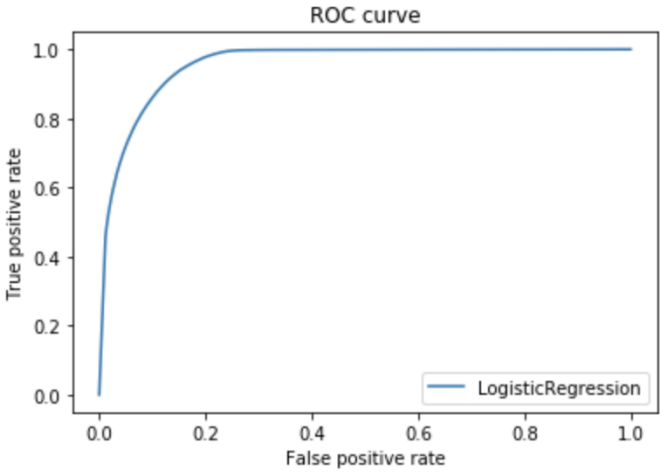


Figure 4: ROC Curve on Train Set. AUC = 0.958079

```
G score is 0.850594
Specificity is 0.934575
Mean accuracy score is 0.901443
Confusion Marix
[[473652  33158]
 [ 29793 102127]]
```

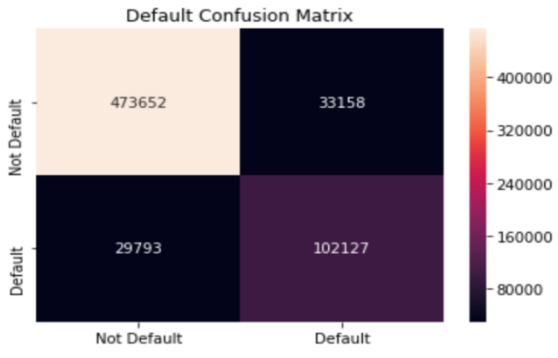


Figure 5: Confusion Matrix on train set.

Extreme Case Test:

We also trained my model by selecting only observation with default loan status and fully paid loan status. We expected to see a 100 percent accuracy rate as the sanity check.

Summary

There are mainly three stages in our project which is roughly following the data science lifecycle displayed on page 4.

The first part is about data processing. We downloaded all the data information from Lending Club's official website and use the python library of panda to combine them with eliminating those cases whose loan status was "current", "Grace period" and "Late." In order to align with the characteristics of our model, we rigorously classified the rest kinds of loan status as binary classes, fully-paid or charged off. Since some features' information are categorical value, we converted them into reasonable numerical value. In terms of zip code information that only shows the first three digits, we tackle them with the substitution by mean and standard deviation of income of certain location based on the first three digits of zip code. For the features with a certain amount of missing data, we set the value of the threshold for the elimination as 20% and we also make sure that the qualify data left over would be sufficient for us on training our model and running the regression test. After this data cleaning and optimization, we split dataset to training set (75%) and test set (25%) and normalized them only based on mean and variance of the training set. Since we assume that the test set should have no knowledge about the mean and variance of its own.

The second part is running the Logistic regression by using the data after the handling of the first part above. In terms of the regularization methods we use, we tried both methodologies of Lasso and Ridge and chose Lasso eventually based on outcomes we got for both methods and it is the method to zero out irrelevant features from the model rather than reduce the impact of irrelevant features by the method of Ridge. After checking the mean accuracy of the dataset, confusion matrix, and ROC curve, we initially notice that the accuracy rate of our prediction model was implausibly high. We then carefully re-investigated our data quality and tried to see if we have some data problem of leakage. Fortunately, we find some features need to be removed and successfully moved into the stage of model validation and parameter tuning. In order to check the usefulness of our model, we also tried to run the dataset from Kaggle and glad to see that the result is pretty close to what we obtained from our latest dataset. With adding the new features from our features engineering, we saw a further improvement in our model as we expected (page 19).

The third stage of our work is trying to implement the method of deep learning. Our model structure has seven layers, which has 64 neurons in the first 4 layers, 32 neurons in the next 2 layers and 16 neurons in the last one layer. And it has 6 dropout layers between each of them. Out layers have 2 neurons for our binary outputs. The optimizer we choose is ADMA with dropout layers since it shows the best performance in our model's optimization. After we accomplished the task of hyper-parameter tuning and model validation, the mean accuracy score of our deep learning model is slightly higher than that of our logistic regression model which is

displayed on page 19. As we mentioned earlier, we particularly remove the cases with the loan status of “current” from our training dataset since it does not give us any information about the final result. We eventually run out deep learning model with these current cases and the model shows that about 18.26 percent of them will be default.

Reference

Ben Luthi. Supermoney. https://www.supermoney.com/2017/09/relief-borrowers-affected-hurricane-harvey/

Ng, Andrew. "Classification." Coursera. https://www.coursera.org/learn/machine-learning/supplement/fDCQp/classification.

Ng., Andrew. "Hypothesis Representation." Coursera. https://www.coursera.org/learn/machine-learning/supplement/AqSH6/hypothesis-representation.

Ng., Andrew. "Gradient Descent." Coursera. https://www.coursera.org/learn/machine-learning/supplement/aEN5G/gradient-descent-for-multiple-variables

Ng. Andrew. "Regularized Logistic Regression." Coursera. https://www.coursera.org/learn/machine-learning/supplement/v51eg/regularized-logistic-regression.

Collins-Thompson., Kevyn. "Model Evaluation and Selection". Coursera. https://www.coursera.org/learn/python-machine-learning/lecture/BE2l9/model-evaluation-selection

"10-Minute Tutorials with Amazon Web Services (AWS)." Amazon Web Services, Inc. Accessed April 28, 2018. https://aws.amazon.com/getting-started/tutorials/.

Brownlee, Jason. "How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras." Machine Learning Mastery. March 10, 2018. Accessed April 28, 2018. https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/.

Chollet, François. Deep Learning with Python. Shelter Island, NY: Manning, 2018. PART 1: THE FUNDAMENTALS OF DEEP LEARNING

DanB. "Data Leakage." Data Leakage | Kaggle. Accessed April 29, 2018. https://www.kaggle.com/dansbecker/data-leakage.

"Deep Learning." Coursera. Accessed April 28, 2018. https://www.coursera.org/specializations/deep-learning.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep Learning. Cambridge, MA: MIT Press, 2017. Ch. 4, 5, 7

James, Gareth. An Introduction to Statistical Learning: With Applications in R. New York: Springer, 2014. Ch. 4, Classification

Ng, Andrew. "Deep Learning." Deep Learning Specialization. Accessed February 28, 2018. https://www.coursera.org/specializations/deep-learning.

Debraj Thakurta, Gary Ericson, Josee Martens, Brad Severtson, Willian Rohm. Microsoft Azure. https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/lifecycle