Measuring Probability of Default Based on Lending Club Historical Data

Yuanhao Feng, Guangyu Qi, Hongbo Cao, Jianing Lyu, Zhiye Yin

Rutgers University

Table of Contents

Measuring Probability of Default Based on Lending Club Historical Data

Introduction

With the increasing accessibility of the internet around the world, Peer-to-Peer Lending has emerged as one of the most popular sources of funding where transactions can be achievable virtually. Individuals can easily seek another individual to support their loans by giving information and posting their needs on Peer-to-Peer Lending websites. Lenders, in return, are able to receive better returns on their investment than traditional banking saving accounts. Lending Club is currently the largest P2P online lending platform (Fabio Caldieraro et al., 2018) and it increasingly provides convenience for more audience and efficiently increases more transactions.

As one of most significant concerns in funding platforms, or in overall financial institutions, the credit risk is the major assessment for both individual investors and the Lending Club so they can ascertain that the individual borrower will pay back the full amount of the loan. In the infant stage of P2P lending, it lacked quantifiable data for researchers and prediction institutions to analyze borrowers' credit risk (Mild Andreas et al., 2015). Currently, there are a plenty of existing researches used both qualitative and quantitative methods to establish model related to credit risk. For example, Lin Xuchen and his fellow researchers combined personal characteristics and loan characteristics to study credit risk in P2P lending in China (Lin X. et al., 2017). However, since our model is primarily based on the historical data from Lending Club, our evidence is constructed upon the quantitative data provided by individual lenders, as we will delete some descriptive data in following univariate analysis section. The main purpose of our project is to help Lending Club to build several models to predict whether a new emerging individual borrower would default by analyzing the historical data provided by Lending Club.

As mentioned above, our purpose is pertaining to the probability of default. Therefore, as one of the most important variables to determine the default probability, Loan Status is our dependent variable with the highest priority to analyze. However, not all data fields, that is, variables in this historical dataset can be used. Therefore, our first step is to exclude variables that cannot relate to the default probability and those that do not make sense to our model. In the following sections, we will list and analyze all variables through discourse analysis in univariate analysis and delete unrelated variables and those with empty fields. This is our first selection of data. Furthermore, we will use histograms to find the most correlated numerical variables to Loan Status result. This is our second selection. Through bivariate analysis, we analyze correlation between variables, compare them, and ascertain final selection of variables that we decide to use in our models: loan_amount, int_rate, dti, revol_util, open_acc, revol_bal, term, emp_length, pub_rec, and recoveries.

Data Analysis

Univariate Analysis

First Selection

Variables that are included in first selection

(1) loan_amount: The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

(2) funded_amnt: The total amount committed to that loan at that point in time.

(3) funded_amnt_inv: The total amount committed by investors for that loan at that point in time.

(4) term(month): The number of payments on the loan. Values are in months and can be either 36 or 60. Its data fields include either “36 month” and “60 month”, and we only kept its numerical means as “36” and “60”. This will be further explained in the following Data Transformation section.

(5) int_rate: Interest Rate on the loan.

(6) installment: The monthly payment owed by the borrower if the loan originates.

(7) grade: LC assigned loan grade. This grade includes from A to G.

(8) sub_grade: LC assigned loan subgrade.

(9) Emp_length: Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years. For variables with “10+”, we included them as “10” groups; for variables “<1”, we transformed it as “0”. This will be further explained in the following Data Transformation section.

(10) home_ownership: The home ownership status provided by the borrower during registration.

(11) annual_inc: The self-reported annual income provided by the borrower during registration.

(12) loan_status: Current status of the loan. As mentioned above, this is our dependent variable, we used “1” as “fully paid” and “0” as “charged off”. This will be further explained in the following Data Transformation section.

(13) dti: A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower’s self-reported monthly income.

(14) delinq_2yrs: The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years.

(15) inq_last_6mths: The number of inquiries in the past 6 months (excluding auto and mortgage inquiries).

(16) open_acc: The number of open credit lines in the borrower's credit file.

(17) pub_rec: Number of derogatory public records.

(18) revol_bal: Total credit revolving balance.

(19) revol_util: Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.

(20) total_acc: The total number of credit lines currently in the borrower's credit file.

(21) recoveries: post charge off gross recovery.

Variables that are excluded in first selection

(1) emp_title: this variable may stand for companies the borrowers worked for. We exclude this data because we cannot make assessment from these companies and we do not have any data to

describe individual company's situations. Therefore, we believe it is unrelated to default probability.

(2) verification_status: Indicates if income was verified by LC, not verified, or if the income source was verified. We exclude this data because it cannot be fit in our model and it is more related to annual_inc, so it has duplicate means of relationship to default probability.

(3) pymnt_plan: this variable has repeated field "n", so we excluded this variable.

(4) desc: Loan description provided by the borrower. This variable is all text (does not have statistical means) and the description is not related to default probability, so we excluded this data.

(5) purpose: A category provided by the borrower for the loan request. This variable is all text (does not have statistical means) and the description is not related to default probability, so we excluded this data.

(6) title: A category provided by the borrower for the loan request. This variable is all text (does not have statistical means) and the description is not related to default probability, so we excluded this data.

(7) zip_code&addr_state: We believe the default probability of a client should be biased based on his/her living locations, so we excluded this data.

(8) mths_since_last_delinq& mths_since_last_record: this data is not complete, so we excluded this data.

(9) initial_list_status& out_prncp& out_prncp_inv: this data includes repeated data fields and does not have statistical means for our model creation, so we excluded this data.

(10) total_pymnt: Payments received to date for total amount funded. This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(11) total_pymnt_inv:Payments received to date for portion of total amount funded by investors. This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(12) total_rec_prncp:Principal received to date: This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(13) total_rec_int: Interest received to date. This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(14) total_rec_late_fee: Late fees received to date. This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(15) collection_recovery_fee: post charge off collection fee. This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(16) last_pymnt_amnt: Last total payment amount received. This data is excluded because it does not make sense since a new emerging client has not yet received a loan, so we excluded this data.

(17) last_pymnt_d&next_pymnt_d&last_credit_pull_d: We believe this data is not related to default probability, so we excluded this data.

(18) application_type: Indicates whether the loan is an individual application or a joint application with two co-borrowers. This variable is all text (does not have statistical means) and the description is not related to default probability, so we excluded this data.

(19) Other remaining variables: those data have empty fields or repeated data, so we excluded them.

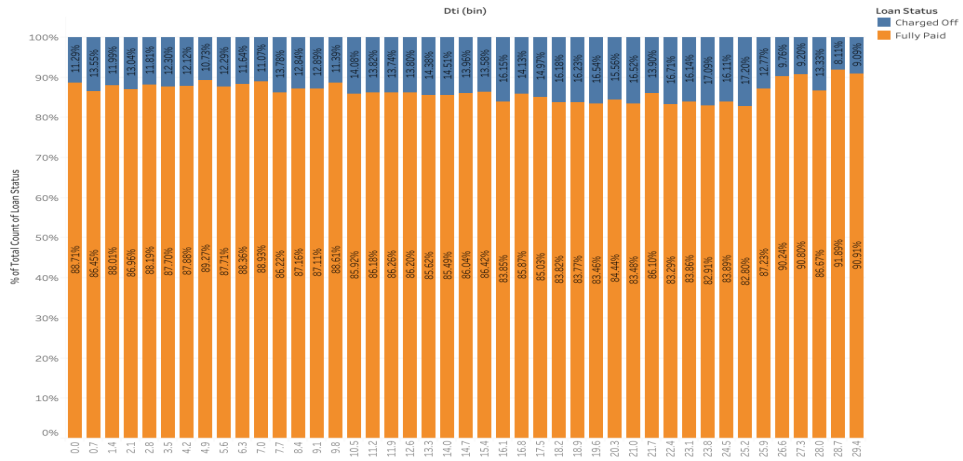
Second Selection

After first selection of variables, we used histograms to analyze if an independent variable has a relatively decent correlation with Loan Status, as our dependent variable. We group the independent variables into bins and take a look at the variation of percentage between "fully paid" and "charged off". If the "charged off" percentage (y-axis) has an overall increasing or decreasing trend corresponding to the increase of the independent variable value (x-axis with separated bins), then we believe it is correlated to Loan Status, therefore, to default probability. After comparison between independent variables, we list those we believe having better correlation and exclude the remaining variables.

Upon our second selection, the remaining 13 variables include: loan_amnt, funded_amnt, funded_amnt_inv, int_rate, annual_inc, dti, open_acc, pub_rec, revol_bal, Term, emp_length

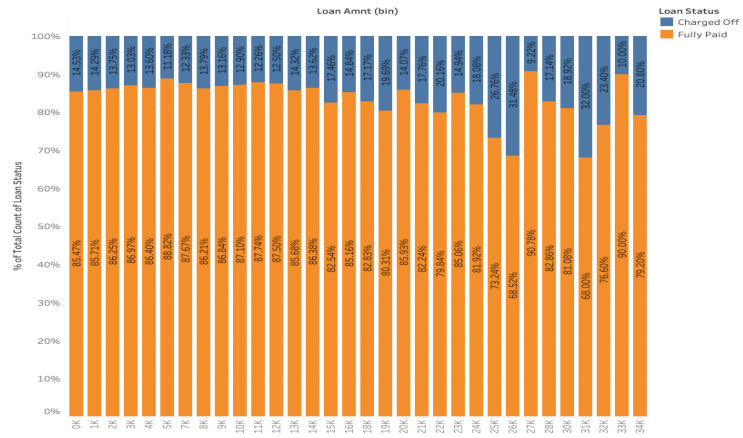
Measuring Probability of Default Based on Lending Club Historical Data

Default Probability by dti bins



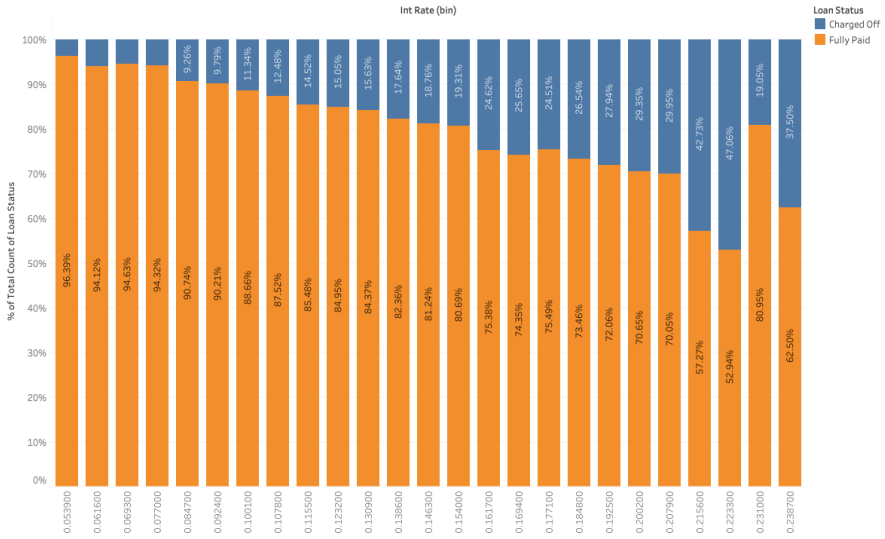
% of Total Count of Loan Status for each Dti (bin). Color shows details about Loan Status. The data is filtered on Exclusions (Installment (bin) 2,Loan Status), which keeps 62 members. Percents are based on each column of each pane of the table.

Default Probability by Loan Amnt bins



% of Total Count of Loan Status for each Loan Amnt (bin). Color shows details about Loan Status. Percents are based on each column of each pane of the table.

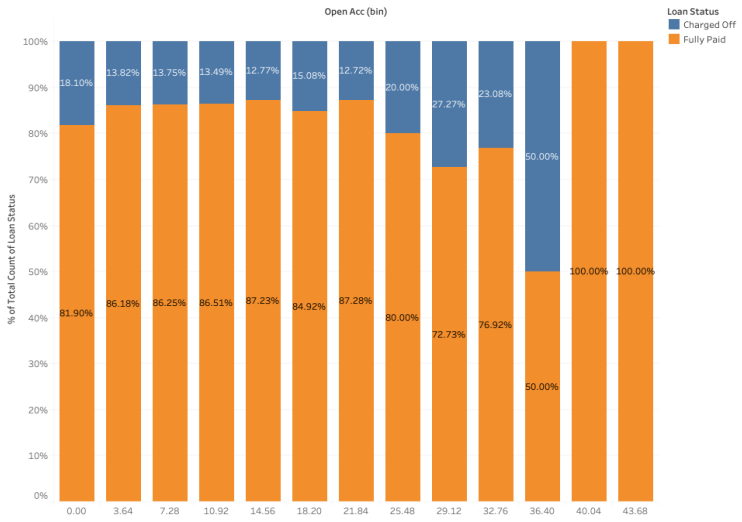
Default Probability by int_rate bins



% of Total Count of Loan Status for each Int Rate (bin). Color shows details about Loan Status. Percents are based on each column of each pane of the table.

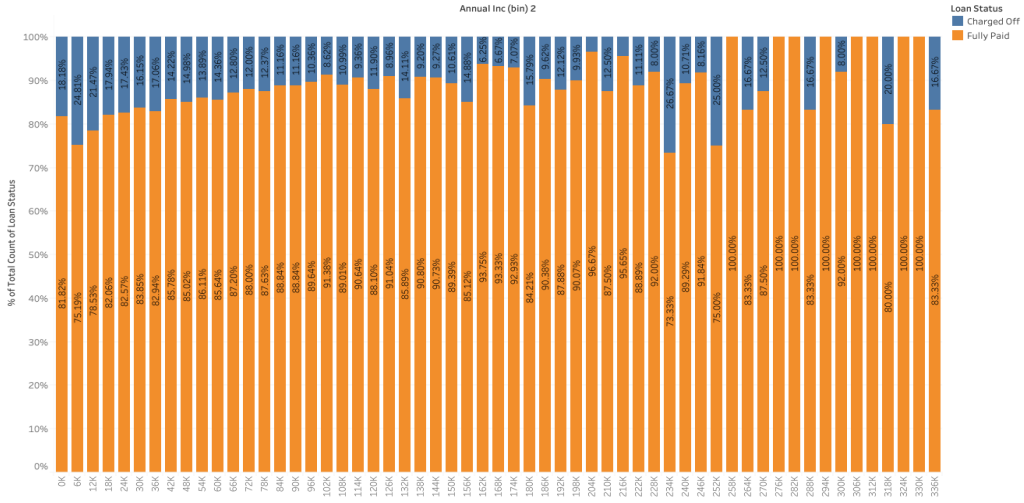
Measuring Probability of Default Based on Lending Club Historical Data

Default Probability by open_acc bins



% of Total Count of Loan Status for each Open Acc (bin). Color shows details about Loan Status. The data is filtered on Exclusions (Installment (bin) 2,Loan Status), which keeps 62 members. Percents are based on each column of each pane of the table.

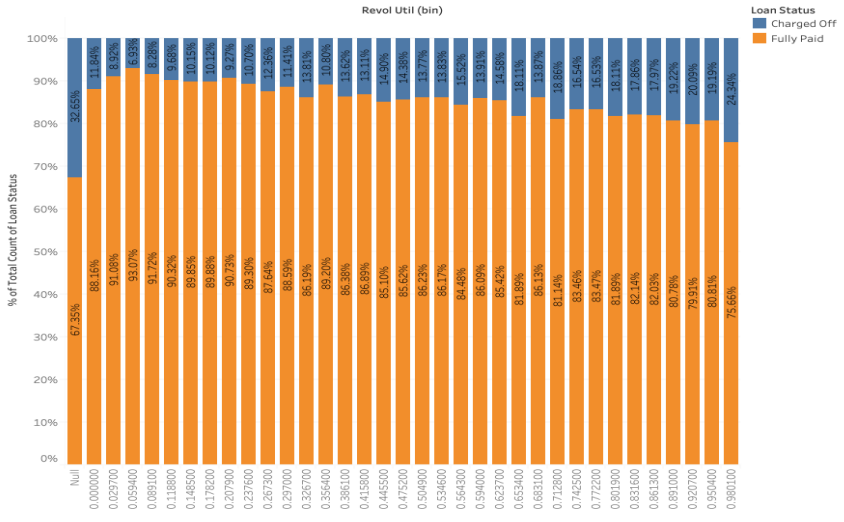
Default Probability by annual_inc bins



% of Total Count of Loan Status for each Annual Inc (bin) 2. Color shows details about Loan Status. The data is filtered on Exclusions (Installment (bin) 2,Loan Status), which keeps 62 members. The view is filtered on Inclusions (Annual Inc (bin) 2,Loan Status), which keeps 106 members. Percents are based on each column of each pane of the table.

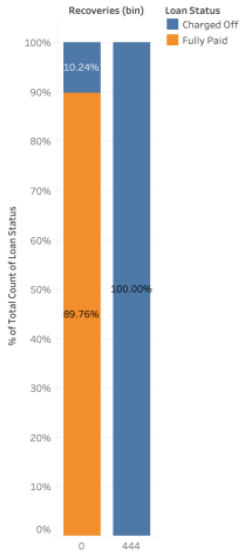
Measuring Probability of Default Based on Lending Club Historical Data

Default Probability by revol_util bins



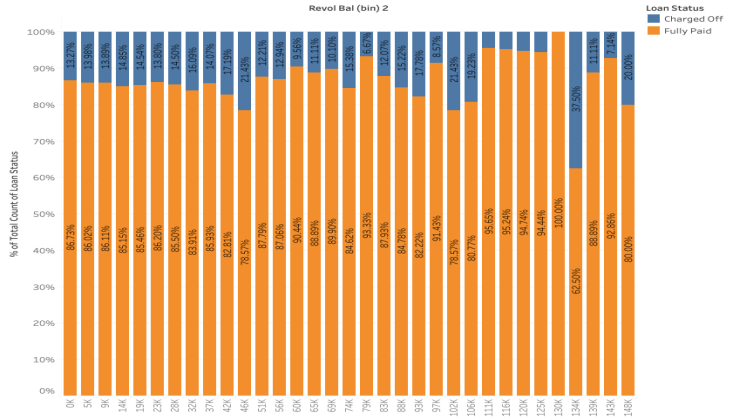
% of Total Count of Loan Status for each Revol Util (bin). Color shows details about Loan Status. Percents are based on each column of each pane of the table.

Default Probability by recoveries bins



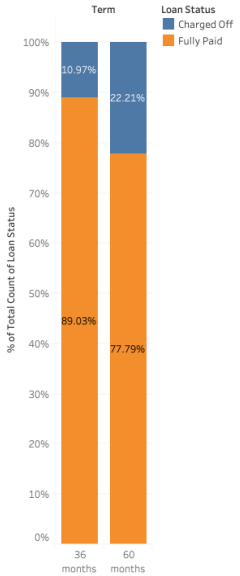
% of Total Count of Loan Status for each Recoveries (bin). Color shows details about Loan Status. The view is filtered on Exclusions (Loan Status,Recoveries (bin)) and Inclusions (Loan Status,Recoveries (bin)). The Exclusions (Loan Status,Recoveries (bin)) filter keeps 40 members. The Inclusions (Loan Status,Recoveries (bin)) filter keeps 3 members. Percents are based on each column of each pane of the table.

Default Probability by revol_bal bins



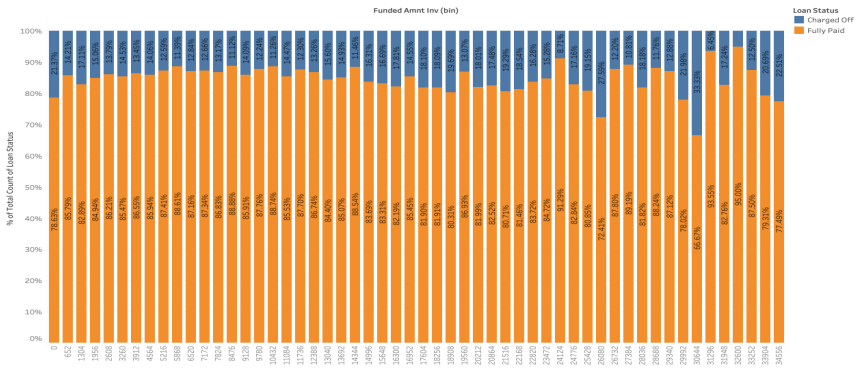
% of Total Count of Loan Status for each Revol Bal (bin) 2. Color shows details about Loan Status. Percents are based on each column of each pane of the table.

Default Probability by term



% of Total Count of Loan Status for each Term. Color shows details about Loan Status. Percents are based on each column of each pane of the table.

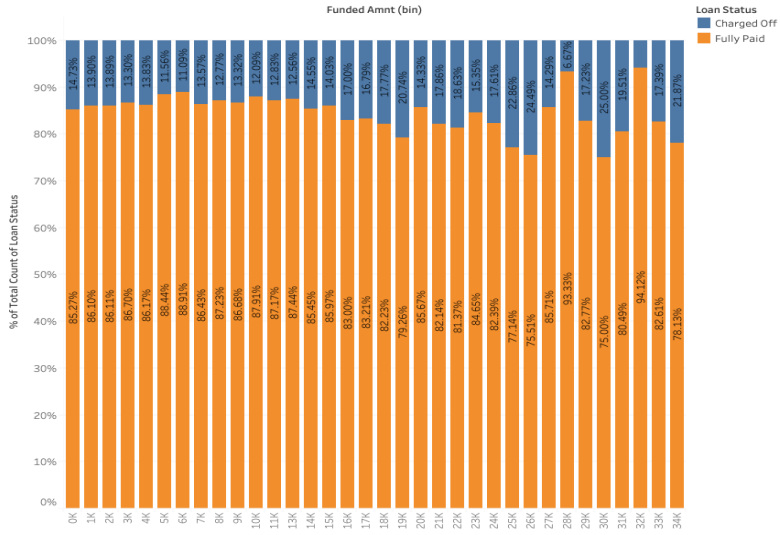
Default Probability by funded_amnt_inv bins



% of Total Count of Loan Status for each Funded Amnt Inv (bin). Color shows details about Loan Status. Percents are based on each column of each pane of the table.

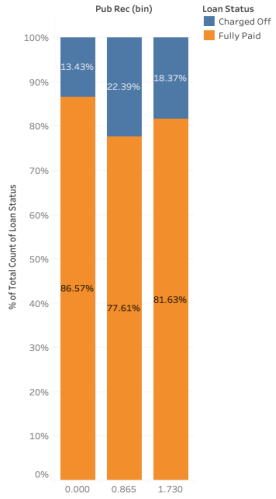
Measuring Probability of Default Based on Lending Club Historical Data

Default Probability by funded_amnt bins



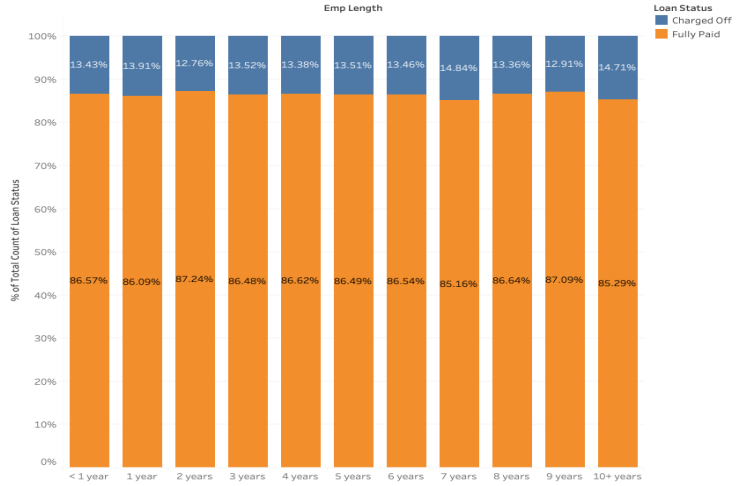
% of Total Count of Loan Status for each Funded Amnt (bin). Color shows details about Loan Status. Percents are based on each column of each pane of the table.

Default Probability by pub_rec bins



% of Total Count of Loan Status for each Pub Rec (bin). Color shows details about Loan Status. The data is filtered on Exclusions (Installment (bin) 2,Loan Status), which keeps 62 members. The view is filtered on Exclusions (Loan Status,Pub Rec (bin)), which keeps 6 members. Percents are based on each column of each pane of the table.

Default Probability by emp_length bins



% of Total Count of Loan Status for each Emp Length. Color shows details about Loan Status. The view is filtered on Emp Length, which excludes n/a. Percents are based on each column of each pane of the table.

Bivariate analysis

Data transformation

Including our dependent variable Loan Status, there are three variables that are categorical variables. Therefore, we have to transform them to numerical form, so we are able to compare correlation between independent variables.

(1) Loan_status

It only includes two categories: “fully paid” and “charged off”. We could represent them by “1” and “0”.

“fully paid” ⇒ “0”

“charged off” ⇒ “1”

(2) emp_length

This variable ranges from “<1”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “>10”. Here we kept numerical data as what it is, and rounded up “<1” to 1 range, and “>10” as “10”.

“<1” ⇒ “0”

“>10” ⇒ “10”

(3) term

This variable includes two categories: “36 month” and “60 month”. We simply delete “month” and keep them as numerical forms.

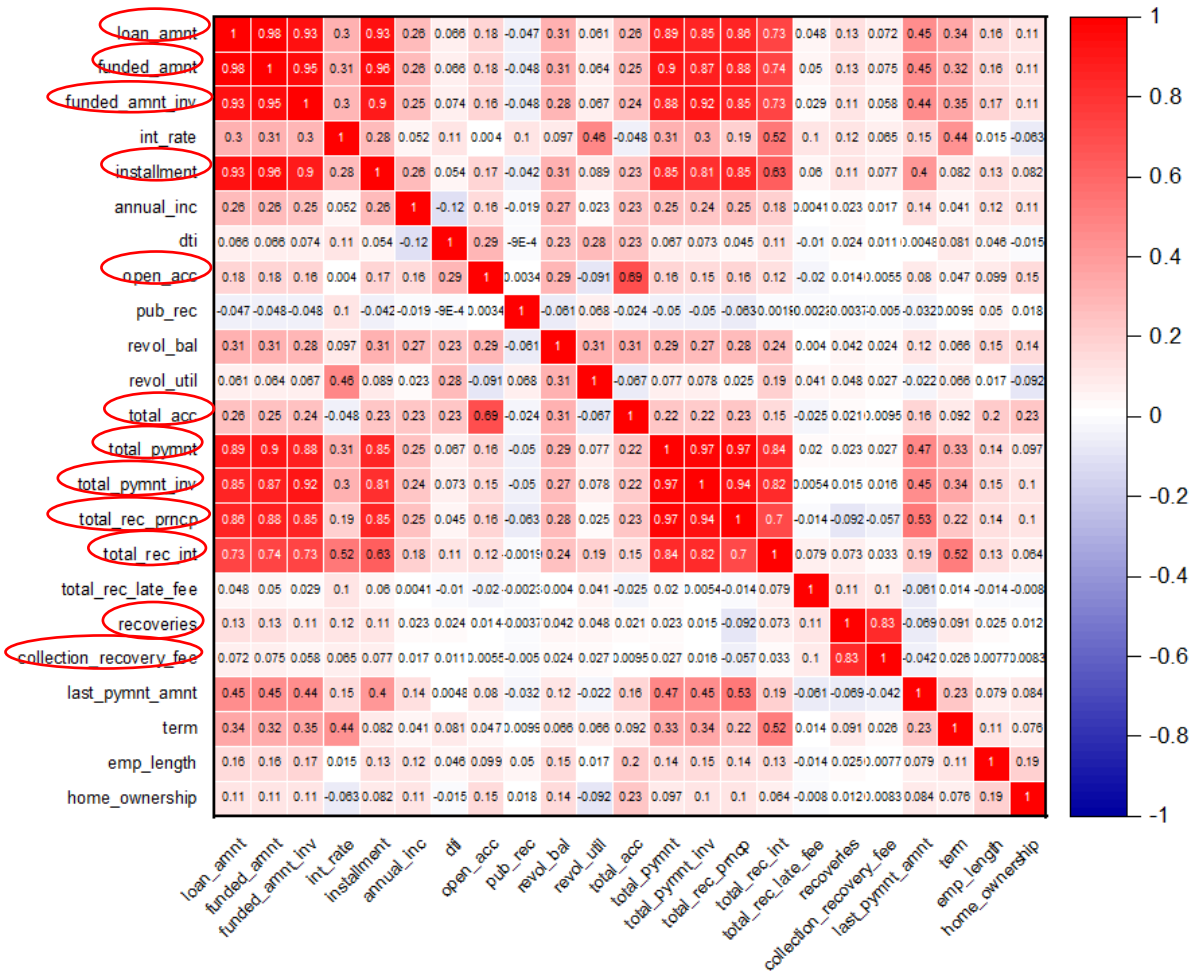
“36 month” ⇒ “36”

“60 month” ⇒ “60”

Measuring Probability of Default Based on Lending Club Historical Data

Variables Selection

Now, we start our bivariate analysis. The purpose of bivariate analysis is to look how high correlations among independent variables affects the accuracy of our model. Firstly, we could have a look at the correlation matrix and mark out the variables with correlation higher than 0.5.



Now, we need see linear relationship between these variables, and we will remove some of them to avoid multicollinearity problem. Depends on the variables' high correlation relationship, we divide them(high correlation variables) into 3 groups to do analysis:

1. Loan_amnt, funded_amnt, funded_amnt_inv, installment, total_pymnt, total_rec_inv, total_rec_prncp, total_rec_int.

2. Open_acc, total_acc.

3. Recoveries, collection_recovery_fee.

Measuring Probability of Default Based on Lending Club Historical Data

(1) Loan_amnt, funded_amnt, funded_amnt_inv, installment, total_pymnt, total_pymnt_inv, total_rec_prncp, total_rec_int.

We build a regression relationship between the eight variables where loan_amnt is y and other variables are x. Regression result below:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.980907
R Square	0.962179
Adjusted R	0.962172
Standard E	1431.018
Observatic	35808

ANOVA

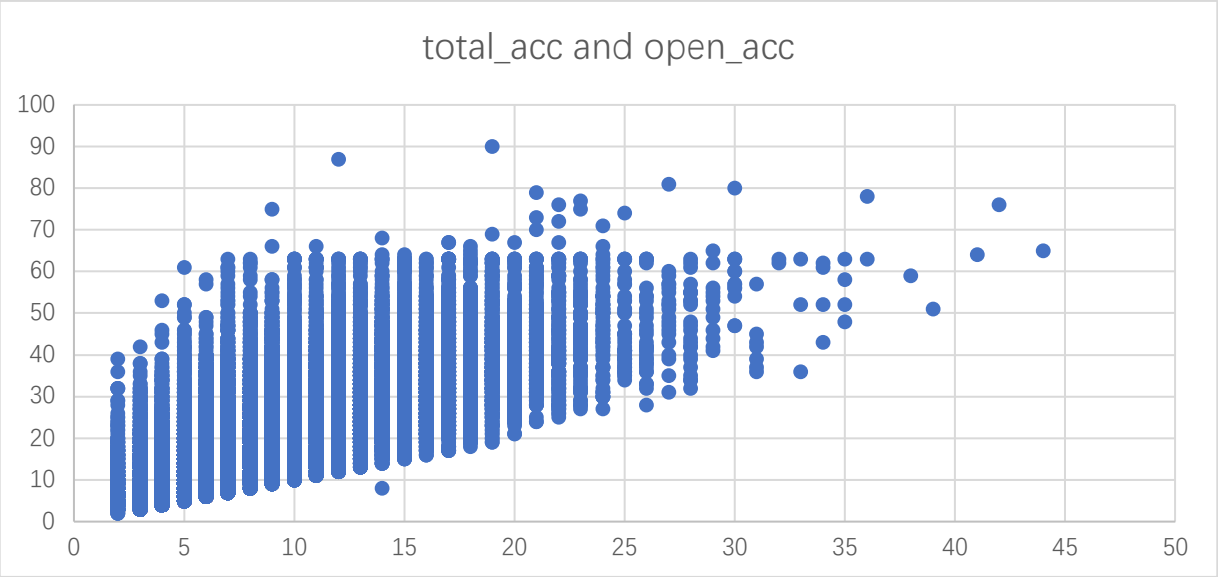
	df	*SS*	*MS*	*F*	*ignificance F*
Regressior	7	1.87E+12	2.66E+11	130109.6	0
Residual	35800	7.33E+10	2047811		
Total	35807	1.94E+12			

	Coefficients	*tandard Err*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	135.0444	14.15846	9.538073	1.54E-21	107.2934	162.7954	107.2934	162.7954
funded_an	1.145687	0.010215	112.1619	0	1.125666	1.165708	1.125666	1.165708
funded_an	-0.01922	0.00989	-1.94338	0.051978	-0.0386	0.000165	-0.0386	0.000165
installmen	-3.75822	0.136693	-27.4939	1.1E-164	-4.02614	-3.4903	-4.02614	-3.4903
total_pym	0.004765	0.014864	0.320561	0.748545	-0.02437	0.033899	-0.02437	0.033899
total_pym	-0.00125	0.009291	-0.13468	0.892864	-0.01946	0.01696	-0.01946	0.01696
total_rec_	-0.00476	0.011714	-0.40611	0.684665	-0.02772	0.018203	-0.02772	0.018203
total_rec_	-0.0083	0.013773	-0.60255	0.546813	-0.03529	0.018696	-0.03529	0.018696

Obviously, linear relationships are between these variables. Especially, we can see the R square of these variables' regression is almost 0.96. Thus, these eight variables are highly correlated. So we only keep **loan_amout** and remove all of other variables.

(2) Open_acc, total_acc.

Measuring Probability of Default Based on Lending Club Historical Data



From the scatter spot above of total_acc and open_acc, we can guess that there may exist linear relationship. To confirm this conjecture, we also need regression:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.689915
R Square	0.475983
Adjusted R	0.475968
Standard E	3.212255
Observatic	35808

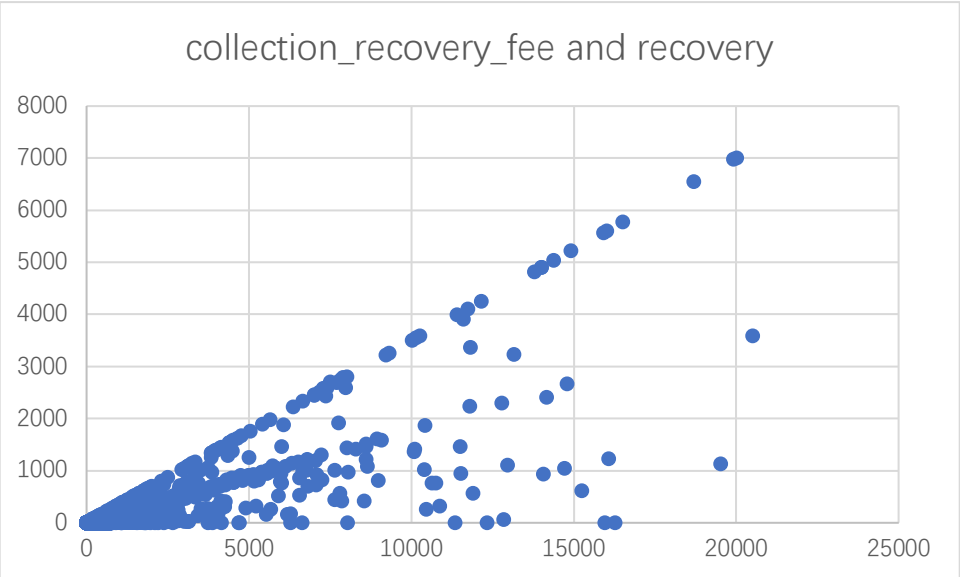
ANOVA

	df	*SS*	*MS*	*F*	*ignificance F*
Regressior	1	335599.7	335599.7	32523.82	0
Residual	35806	369467.1	10.31858		
Total	35807	705066.8			

	Coefficients	*tandard Erı*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	3.380022	0.036856	91.70921	0	3.307784	3.452261	3.307784	3.452261
total_acc	0.266683	0.001479	180.3436	0	0.263784	0.269581	0.263784	0.269581

Obviously, linear relationships are between these variables. Especially, we can see the R square of these variables' regression is almost 0.48 and P-value is equal to zero. Thus, these two variables are highly correlated. So we only keep open_acc and remove total_acc.

(3) Recoveries, collection_recovery_fee.



From the scatter spot above collection_recovery_fee and recovery, we can guess that there may exist linear relationship. To confirm this conjecture, we also need regression:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.830391
R Square	0.689549
Adjusted R	0.689541
Standard E	374.8566
Observatic	35808

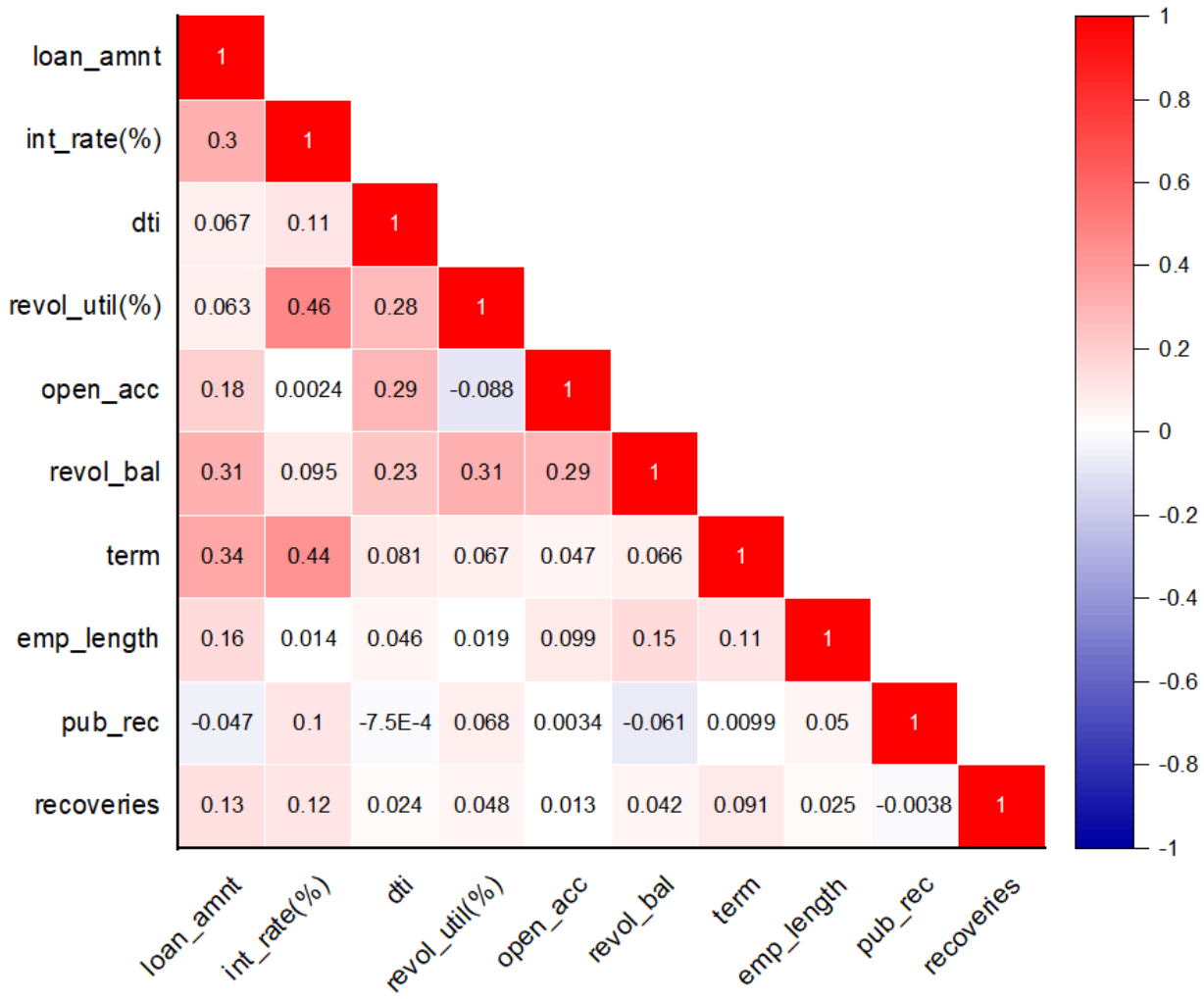
ANOVA

	df	*SS*	*MS*	*F*	*ignificance F*
Regressior	1	1.12E+10	1.12E+10	79529.56	0
Residual	35806	5.03E+09	140517.5		
Total	35807	1.62E+10			

	Coefficients	*tandard Erı*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	45.41617	1.987829	22.84712	1E-114	41.51997	49.31238	41.51997	49.31238
collection_	3.606347	0.012788	282.0098	0	3.581282	3.631412	3.581282	3.631412

Obviously, linear relationships are between these variables. Especially, we can see the R square of these variables' regression is almost 0.69 and P-value is equal to zero or close to zero. Thus, these two variables are highly correlated. Therefore, we only keep recovery and remove collection_recovery_fee.

To remove those variables we don't need, we also pick other 7 variables to test our model, and the correlation between all these 10 variables are look like this:



Now, we can start our modeling work.

Models Creation

Totally, we created five different models using python:

- Logistic regression
- Decision Tree
- K-Nearest Neighbors
- Neural Network
- Random Forest

Confusion matrix and CAP curve are our major methods to do the performance test on both training sample and test sample. For confusion matrix, we used built-in confusion_matrix function. Below is our function for CAP curve plotting.

In [38]:
```
#model performance
#CAP curve function
from matplotlib import cm
from scipy import integrate
def capcurve(y_values, y_proba):
    num_sum = np.sum(y_values)
    num_count = len(y_values)
    rate_val = float(num_sum) / float(num_count)
    ideal = pd.DataFrame({'x':[0,rate_val,1],'y':[0,1,1]})
    xx = np.arange(num_count) / float(num_count - 1)

    y_cap = np.c_[y_values,y_proba]
    y_cap_df = pd.DataFrame(data=y_cap)

    y_cap_df = y_cap_df.sort_values([1], ascending=False)
    y_cap_df = y_cap_df.reset_index(drop=True)

    yy = np.cumsum(y_cap_df[0]) / float(num_sum)
    yy = np.append([0], yy[0:num_count-1])

    percent = 0.5
    row_index = np.trunc(num_count * percent)
    row_index = row_index.astype(np.int32)

    sigma_perfect = 1 * xx[num_sum - 1 ] / 2 + (xx[num_count - 1] - xx[num_sum]) * 1
    sigma_model = integrate.simps(yy,xx)
    sigma_random = integrate.simps(xx,xx)

    gini_value = (sigma_model - sigma_random) / (sigma_perfect - sigma_random)

    fig, ax = plt.subplots(nrows = 1, ncols = 1)
    ax.plot(ideal['x'],ideal['y'], color='grey', label='Perfect Model')
    ax.plot(xx,yy, color='b', label='User Model')
    ax.plot(xx,xx, linestyle='dashed', color='r', label='Random Model')

    plt.xlim(0, 1.0)
    plt.ylim(0, 1.05)
    plt.title("CAP Curve. AR ="+str(np.round(gini_value,4)))
    plt.xlabel('% of the data')
    plt.ylabel('% of bad')
    plt.grid(True)
    plt.legend(loc = 'lower right')
    plt.savefig('CAP.png')
    plt.show()
```

Logistic Regression

Principle

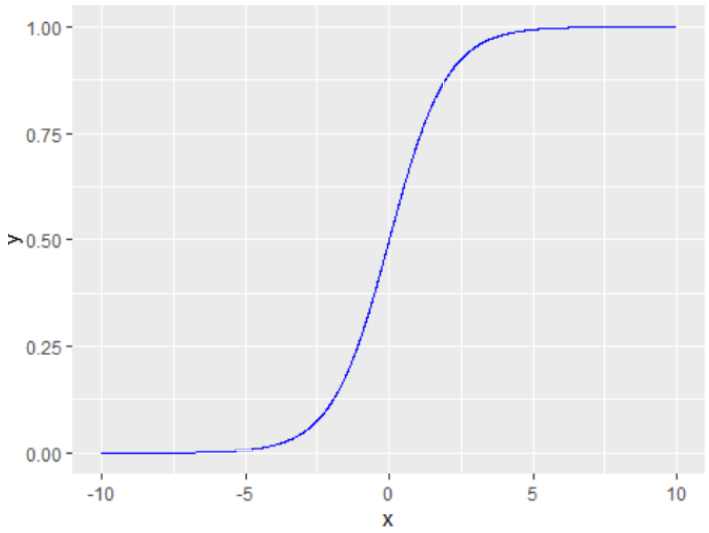
Logistic regression is mainly used to solve the binary classification problem, and its output is 0 and 1 dependent variables.

Logistic regression measures the relationship between the dependent variable and one or more independent variable characteristics by estimating the probability using its inherent logistic function. These probabilities must then be binarized to truly predict. This is the task of the logistic function, also known as the sigmoid function. The Sigmoid function is a sigmoid curve. It can map any real value to a

value between 0 and 1, but it does not take 0 or 1. Then we need to use a threshold classifier to convert values between 0 and 1 to 0 or 1.The important formula of Logistic is to establish a fitting relationship between features and category probability:

$$f(x) = \frac{1}{1 + e^{-x}}$$

This following picture shows off how the logistic regression chart looks like and how the visualized relationship would be:



Our goat is to maximize the probability of random data points being precisely classified (the MAX likelihood estimation). the MAX likelihood estimation is a general tool for estimating those picked parameters in statistical models.

Measuring Probability of Default Based on Lending Club Historical Data

Behind the Scene Logic:

There are n independent variables for the response variable y, denoted as x1, x2, ... , xn. The conditional probability of success under the action of n independent variables is recorded as P = P(y = 1|x1, x2, ... , xn) . Then the logistic regression model is：

$$P = \frac{\exp\left(\beta_0 + \beta_1 x_1 + \beta_2\, x_2 + \cdots + \beta_n\, x_n\right)}{1 + \exp\left(\beta_0 + \beta_1 x_1 + \beta_2\, x_2 + \cdots + \beta_n\, x_n\right)}$$

Where β0 is called a constant term or intercept, and β1, β2, ... , βn is called a logistic model regression coefficient.

When we use Mathematical way to “log”-transform this formula, we can obtain the following linear equation:

$$logit(P) = \ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x_1 + \beta_2\, x_2 + \cdots + \beta_n\, x_n$$

Model Specification

Firstly, we combine the training dataset with the testing dataset in order to increase the size of our dataset and boost the randomization. For the logistic regression model, we eventually decide to use 10 relatively independent variables to build up the model.

Secondly, since we combine these two datasets in the first place, here in our code, we import train-test split from sklearn package to split our dataset by 9:1: 90% is training dataset, 10% is testing dataset.

Thirdly, we import logistic regression model package to deal with the dataset we set up; For the logistic regression model, the highly important part is to choose the suitable parameters. In this case, we tried to

use GridSearchCV to help us to define those fitting parameters. Here is the picture of what we got from it:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='liblinear',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [0.1, 0.3, 1, 3, 10, 30, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(recall_score), verbose=0)
```

Here's one thing we have to notice is that param_grid 'C' is what we initially made it random by numbers, we have to figure out and pick up the best param_grid 'C' for the model, which needs to use logistic search of best parameters:

```
logit_search.best_params_
```

```
{'C': 100}
```

Therefore, now we can run the code up to turn out the result of the logistic regression model:

	precision	recall	f1-score	support
0	0.97	1.00	0.99	3393
1	1.00	0.82	0.90	586
accuracy			0.97	3979
macro avg	0.99	0.91	0.94	3979
weighted avg	0.97	0.97	0.97	3979

Here's the result we received finally; precision of 1 means that the accuracy of the prediction of people who would be charged off (default), and precision of 0 means that the accuracy of the prediction of people who would be fully paid (no default).

Oversampling

Notice: Since the original dataset is highly unbalanced, we need to use the tool named oversampling (SMOTE) to balance up the dataset. The reason why we surly need to do so is because for logistic regression model, if the dataset is very unbalanced, the statics it turns out will more trend to the side, the higher percentage of the y-variable in the dataset, in this case, the final result would be certainly less fitting and precise. Back to our model, since in our dataset, the percentage of "0" fully paid is extremely higher than the percentage of "1" charged off; So we have two methods to balance them up, the first is to increase the size of "1" charged off, the second is to decrease the size of "0" fully paid. We picked the first option: increasing the size of "1" charged off.

Here in Python, we import SMOTE from imlearn.over_sampling package. We set up "1" charged off as a bad data so that we can change its size by following step:

```
minor_size = int(bad_size * 0.9)
non_bad_data = train_data[train_data['loan_status']==0].sample(n=minor_size, random_state=999,replace = True)
SMOTE_data = pd.concat([bad_data,non_bad_data])
SMOTE_x = SMOTE_data.drop('loan_status', axis = 1)
SMOTE_y = SMOTE_data['loan_status']
```

In the the second line of code the above pic shows off, we tried to match the size of "1" charged off with the size of "0" fully paid. After we finished this part, we need to swap the data that will be plugged into the logistic regression model:

```
SMOTE_x_train = SMOTE_x
SMOTE_x_test = x_test
SMOTE_y_train = SMOTE_y
SMOTE_y_test = y_test
```

Competed all the steps for oversampling we should process, now we can repeat the logistic regression model to figure out the more accurate result:

	precision	recall	f1-score	support
0	0.97	1.00	0.99	3393
1	1.00	0.85	0.92	586
accuracy			0.98	3979
macro avg	0.99	0.92	0.95	3979
weighted avg	0.98	0.98	0.98	3979

Here it might raise up another concern: why the precision accuracy looks like the same in both scenarios. The reasons we conducted are: Firstly, we set up the model to retain only 2-digit decimals; Secondly, the variables we picked up are highly fitted for the model so that the result would be great.

Model Performance

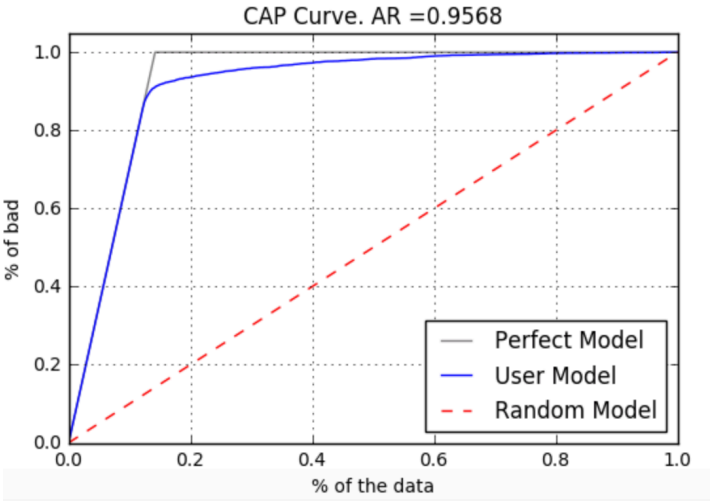
(1) In the Sample

Confusion Matrix:

	0	1
0	30723	0
1	1061	4023

In the confusion matrix, the true positive is 30723 and the true negative is 4023. The false positive is 0 and the false negative is 52. The accuracy is (30723+4023)/(35807) = 97.04% which is a awesome performance.

CAP curve:



The AR for logistic regression model is 0.9568 which is a great performance.

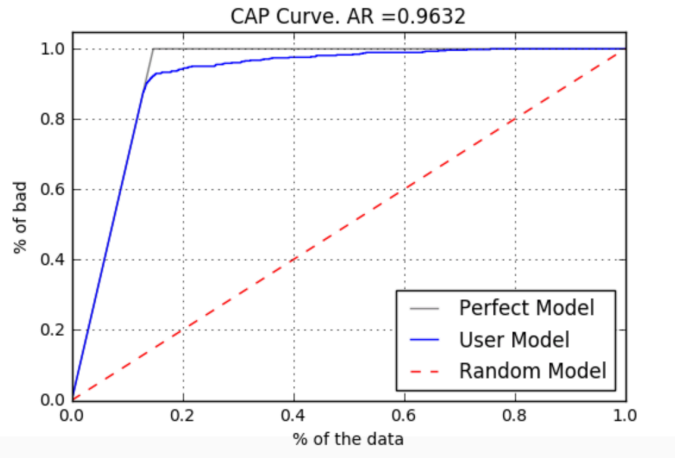
(2) Out of the Sample

Confusion Matrix:

	0	1
0	3393	0
1	103	483

In the confusion matrix, the true positive is 3393 and the true negative is 483. The false positive is 0 and the false negative is 103. The accuracy is (3393+483)/(3979) = 97.41% which is also an amazing performance.

CAP curve:



For out of sample test, we still got an 96.32% AR which is an excellent performance.

Pros and Cons

Advantages:

- Easy to implement, interpret and efficient to train
- Fast classifying speed
- Less inclined to over-fitting

Disadvantages:

- Tough to obtain complex relationships
- Requires average or no multicollinearity between variables
- Assume linearity between dependent variable and independent variables

Measuring Probability of Default Based on Lending Club Historical Data

Decision Tree

Principle

A decision tree is an analysis method that determines whether the next prediction point will occur based on the known probability of various situations. A decision tree is essentially a summary of past objective conclusions based on past feature information. Decision trees are divided into two categories: classification trees and regression trees. The classification tree can classify data sets whose target features are classification types, such as yes or no. Regression trees are used for data sets whose target features are numeric, including continuous or discrete values, such as stock prices.

First of all, the main thing of decision tree is to select features to divide the data set, so it is necessary to find the features that are most proper one for splitting by ratio. There are three primary algorithms for decision trees, including three different objective functions that can be used to select the features: the first one: ID3 algorithm uses information gain as the node splitting index; the second one: C4.5 algorithm uses information gain rate as the node splitting index; the third one: CART algorithm uses

Gini index as the index of node splitting. Second of all, thanks to the purity judgment method, the optimal splitting point would be found. And the data is split into two parts with the higher purity based on this feature. The purity judgment method means that when the decision tree is generated, the data of each branch is expected to belong to the same category as much as possible. Then it is judged whether the requirements are matched--- if it fails to meet the requirements, it repeatedly finds suitable splitting characteristics to continue splitting until it meets the requirements. The last but not least, the decision tree is pruned to prevent the over-fitting of the decision tree, which has two methods: pre-pruning and post-pruning.

Here we mainly talk about the the third objective function "CART algorithm". The CART algorithm can generate classification trees and regression trees. The CART algorithm uses the Gini index to select appropriate features for splitting. The Gini index is used to measure the probability that two random items belong to the same category. The Gini index of each node is equal tot he sum of the squares of the proportion of instance data in each category under the node. To calculate the current Gini index, the formula is as following picture shown:

$$\text{Gini(p)} = \Sigma_{k=1}^{K} p_k(1 - p_k) = 1 - \Sigma_{k=1}^{K} p_k^2$$

For the dataset we handle, we need to assume that there are K classifications, the probability that the sample belongs to the kth class is pk (means every class has a matched probability).

So the first step we need to do is to find the best division in regard to the Gini index. The method is to traverse the features of all the samples, obtain the features that can make the Gini index change the most before and after the division, and divide the tree into left and right sub-trees according to the values of the features. Then continue to divide the left and right sub-trees by repeating the same pattern.

Then, we should determine whether the Gini index currently divided is 0 or not. If it turns out that it's not 0, we have to repeat the above division of processing for the left and right sub-trees until the Gini index gets 0, where the whole process stops dividing.

Model Specification

Here's another thing we need to deal with before we start to build up our model--- we need to use the tool "scaling" to make all the variables we picked up become normalized (satisfied normal distribution). In our code, we import StandardScaler from sklearn.preprocessing, which can usefully make all the variables normalized:

```
full_data['trans_loan_amnt']=StandardScaler().fit_transform(full_data['loan_amnt'].values.reshape(-1,1))
full_data['trans_int_rate']=StandardScaler().fit_transform(full_data['int_rate'].values.reshape(-1,1))
full_data['trans_revol_util']=StandardScaler().fit_transform(full_data['revol_util'].values.reshape(-1,1))
full_data['trans_open_acc']=StandardScaler().fit_transform(full_data['open_acc'].values.reshape(-1,1))
full_data['trans_dti']=StandardScaler().fit_transform(full_data['dti'].values.reshape(-1,1))
full_data['trans_revol_bal']=StandardScaler().fit_transform(full_data['revol_bal'].values.reshape(-1,1))
full_data['trans_term']=StandardScaler().fit_transform(full_data['term'].values.reshape(-1,1))
full_data['trans_emp_length']=StandardScaler().fit_transform(full_data['emp_length'].values.reshape(-1,1))
full_data['trans_pub_rec']=StandardScaler().fit_transform(full_data['pub_rec'].values.reshape(-1,1))
full_data['trans_recoveries']=StandardScaler().fit_transform(full_data['recoveries'].values.reshape(-1,1))
```

In the code, we rename all the variables by adding "trans" in front of their original names to remind us that they are transferred into the standard of normal distribution.

So now we can start to deal with setting up the decision tree model: firstly, we combine the training dataset with the testing dataset in order to increase the size of our dataset and boost the randomization. For the decision tree model, we eventually decide to use 10 relatively independent variables to build up the model. Secondly, since we combine these two datasets in the first place, here in our code, we import train-test split from sklearn package to split our dataset by 9:1: 90% is training dataset, 10% is testing dataset. Thirdly, we import decision tree model package of Python to deal with the dataset we set up;

For the decision tree model, the highly important part is to choose the suitable parameters as well, parameters are really the keys of the success to determine we will have a good-performance model or a bad one. In this case, we tried to use DecisionTreeClassifier and RandomizedSearchCV to help us to define those fitting parameters. Here is the picture shows off how we got from it:

```
Decitree = DecisionTreeClassifier()

Decitree_param = {"criterion": ["gini", "entropy"],
                  "max_depth": [3, None],
                 "max_features": sp.stats.randint(1, 6),
                 "min_samples_split": sp.stats.randint(2, 14),
                 }

Decitree_search = RandomizedSearchCV(Decitree, param_distributions = Decitree_param,  scoring = recall_score)
```

In details, we need to point out that the parameter named max_depth represents the maximum depth of the decision tree, which is the most significant parameter affecting the whole model. Therefore, it is truly necessary to find out the optimal maximum depth parameter. We actually don't have any smart way to solve this issue, only thing we did is to try as many times as we can. So we first tried to put 1, 2, and 3 for max_depth, it turned out that the accuracy actually was increased as we added up the number; However, when we tried to put 4, 5, and 6 for max_depth, which turns out that the accuracy was slightly decreased as we added up the number. We thought that we might figure out the best number for this parameter, but we need to confirm this assumption. Therefore, we increased later-tried numbers by 5 started from 3 like 3, 8, 13, 18 ,etc.Finally, we confirmed our assumption: the accuracy values would be decreased as the maximum depth value of the parameter less increased, and the accuracy values would reach the maximum as the maximum depth value of the parameter is 3. Therefore, the optimal maximum depth value of 3 is selected for model fitting.

```
{'criterion': 'gini',
 'max_depth': None,
 'max_features': 3,
 'min_samples_split': 4}
```

Therefore, now we can run the code up to turn out the result of the decision tree model:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3393
1	0.99	0.99	0.99	586
accuracy			1.00	3979
macro avg	1.00	0.99	0.99	3979
weighted avg	1.00	1.00	1.00	3979

Here's the result we received finally; precision of 1 means that the accuracy of the prediction of people who would be charged off (default), and precision of 0 means that the accuracy of the prediction of people who would be fully paid (no default).

Oversampling

Notice: As we mentioned before in the logistic regression model, since the original dataset is highly unbalanced, we need to use the tool named oversampling (SMOTE) to balance up the dataset. The reason why we surly need to do so is we need to pursue the precision of the whole project. No matter what model we use, every model existed in the world has its own error range systematically and non-systematically. Back to our model, since in our dataset, the percentage of "0" fully paid is extremely higher than the percentage of "1" charged off; So we have two methods to balance them up, the first is to increase the size of "1" charged off, the second is to decrease the size of "0" fully paid. We picked the first option: increasing the size of "1" charged off.

Here in Python, we import SMOTE from imlearn.over_sampling package. We set up “1” charged off as a bad data so that we can change its size by following step:

```
minor_size = int(bad_size * 0.9)
non_bad_data = train_data[train_data['loan_status']==0].sample(n=minor_size, random_state=999,replace = True)
SMOTE_data = pd.concat([bad_data,non_bad_data])
SMOTE_x = SMOTE_data.drop('loan_status', axis = 1)
SMOTE_y = SMOTE_data['loan_status']
```

In the the second line of code the above pic shows off, we tried to match the size of “1” charged off with the size of “0” fully paid. After we finished this part, we need to swap the data that will be plugged into the decision tree model:

```
SMOTE_x_train = SMOTE_x
SMOTE_x_test = x_test
SMOTE_y_train = SMOTE_y
SMOTE_y_test = y_test
```

After all, we can activate the part that products the oversampling decision tree model, Here are the two pictures, one shown the code, another one shown the result:

```
SMOTE_logit = GridSearchCV(LogisticRegression(solver = 'liblinear'
SMOTE_logit.fit(SMOTE_x_train,SMOTE_y_train)
SMOTE_logit_predictions = SMOTE_logit.predict(SMOTE_x_test)
SMOTE_logit_prob = SMOTE_logit.predict_proba(SMOTE_x_test)[:,1]
```

	precision	recall	f1-score	support
0	0.97	1.00	0.99	3393
1	1.00	0.85	0.92	586
accuracy			0.98	3979
macro avg	0.99	0.92	0.95	3979
weighted avg	0.98	0.98	0.98	3979

Here it might be surprisingly concern: why the both ways (“0” fully paid and “1” charged off) of precision accuracy is very high, almost approach to the perfection. The reasons we conducted are: Firstly, using decision tree model for this dataset is meaningful, that’s been saying decision tree model could perform well in unbalanced dataset; Secondly, the variables we picked up are highly fitted so that the result would be excellent.

Model Performance

(1) In the Sample

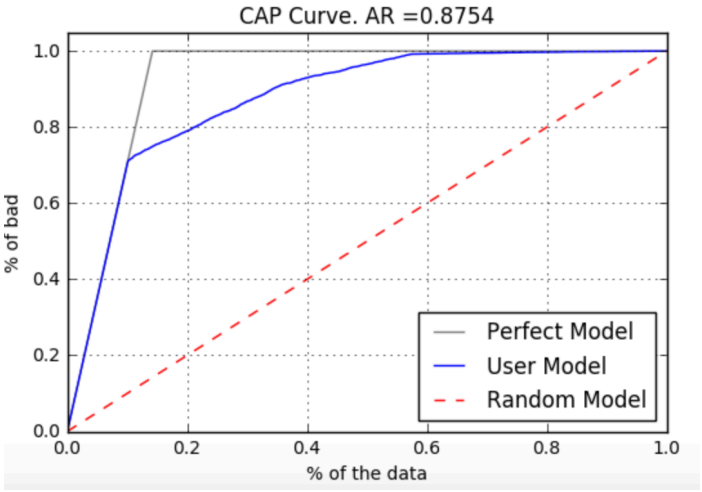
Measuring Probability of Default Based on Lending Club Historical Data

Confusion Matrix:

	0	1
0	30723	0
1	1471	3613

In the confusion matrix, the true positive is 30723 and the true negative is 3613. The false positive is 0 and the false negative is 1471. The accuracy is (30723+3613)/(35807) = 95.89% which is a great performance.

CAP curve:



The AR for decision tree model is 87.54%. It is a little lower than logistic regression model.

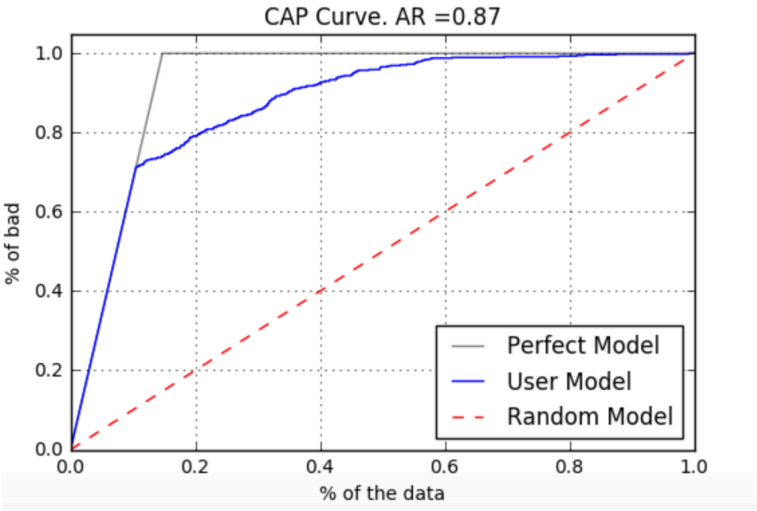
(3) Out of the Sample

Confusion Matrix:

	0	1
0	3393	0
1	169	417

In the confusion matrix, the true positive is 3393 and the true negative is 417. The false positive is 0 and the false negative is 169. The accuracy is (3393+417)/(3979) = 95.75% which is also an amazing performance.

CAP curve:



For out of sample test, we still got an 87% AR which is great but not a better one than logistic model.

Pros and Cons

Advantages:

- Easy to explain and require less effort for data preparation
- Missing values do NOT affect the process of building a decision tree to any considerable extent.

Disadvantages:

- Have better performance for small data sample
- Relatively expensive as the complexity and time have taken are more

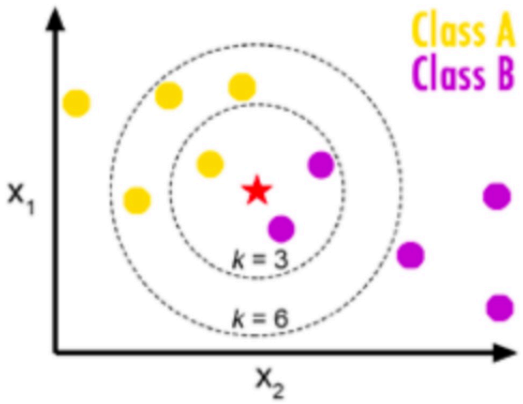
K-Nearest Neighbors

Principle

K-nearest neighbors algorithm is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, normalizing the training data can improve its accuracy dramatically. It is simple, easy-to-implement supervised. K-nearest neighbors algorithm is as known as k-NN.

The function is only approximated locally and all computation is deferred until function evaluation, and it can be use to solve both classification and regression problems, satisfied our project, which is a classification project.

The k-NN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. So it utilize the formula of distance between two points.



In the graph shown above, we first pick a value for k, such as in the inner circle: k=3; in the outer circle: k=6. Then we take k nearest neighbors of the new data point according to their distance. Among these neighbors, count the number of data points in each category and assign the new data point to the

category where you counted the most neighbors. Like the inner circle with k=3 has 1 yellow and 2 purples, and the outer circle with k=6 has 4 yellows and 2 purples. (Yellow in Class A, Purple in Class B)

With the formula shown a dataset where x is a matrix of features from an observation and y is a class label. With a positive integer k, *k*-NN looks at k observations closest to the test observation x(0), also estimates the conditional probability that it belongs to class j.

$$P\left(y = j|\mathbf{x} = \mathbf{x_0}\right) \approx \frac{1}{k} \sum_{i \in \mathcal{N}} \mathbf{I}\{y^{(i)} = j\}.$$

K-NN Algorithm

Step 1: Load the data and pick a positive integer value for k.

Step 2: Take the *k* nearest neighbors of the new data point according to their distance.

Step 3: Among these neighbors, count the number of data points in each category and assign the new data point to the category where you counted the most neighbors.

Proper Value for k

To select the k that's right for the data, we run the k-NN algorithm several times with different values of k and choose a k that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

(1) As we decrease the value of k to 1, our predictions become less stable.

(2) Inversely, as we increase the value of k, our predictions become more stable due to majority averaging, and thus, more likely to make more accurate predictions (up to a certain point).

(3) However, if we choose a very big k value, the number of errors will increase because the scale we choose is too large.

Finally, we choose the value of k equals to 40.

Model Specification

In our project, we utilized KNeighborsClassifier() from sklearn.neighbors which contains 8 hyperparameters.

We try to find the optimal value of k and utilize it in our model:

```
best_score = 0.0
best_k = -1
for k in range(1, 100) :
    knn_clf = KNeighborsClassifier(n_neighbors=k)
    knn_clf.fit(X_train, y_train)
    score = knn_clf.score(x_test, y_test)
    if score > best_score:
        best_k = k
        best_score = score

print("best_k = " + str(best_k))
print("best_score = " + str(best_score))
```

With the options between ‘uniform’ and ‘distance’ to find the best value of k:

```
from sklearn.model_selection import GridSearchCV

params_grid = [
    {
        'weights':['uniform'],
        'n_neighbors':[i for i in range(1, 100)]
    },
    {
        'weights':['distance'],
        'n_neighbors':[i for i in range(1, 100)],
        'p':[i for i in range(1, 6)]
    }
]

knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, params_grid)
grid_search.fit(x, y)
```

Finally, we use k = 40, as well as n_neighbors = 40 in KNeighborsClassifier().

```
[56]: #KNN
from sklearn.neighbors import KNeighborsClassifier
KNeighbors = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
 metric_params=None, n_jobs=1, n_neighbors=40, p=2,
 weights='uniform')
KNeighbors.fit(SMOTE_x_train,SMOTE_y_train)
```

Hyperparameters for KNeighborsClassifier

Hyperparameters	Explanation	Input
algorithm	algorithm used to compute the nearest neighbors	default
leaf_size	leaf size passed to BallTree or KDTree	default
metric	the distance metric to use for the tree	default
metric_params	additional keyword arguments for the metric function	default
n_jobs	the number of parallel jobs to run for neighbors search	default
n_neighbors	number of neighbors to use	40

p	power parameter for the Minkowski metric	default
weights	weight function used in prediction	default

The graph below shows our result for K-nearest neighbors model.

```
[59]: #KNN model result
      print(classification_report(y_test,KNN_predictions))
```

```
              precision    recall  f1-score   support

           0       0.92      0.80      0.86      3393
           1       0.34      0.61      0.44       586

    accuracy                           0.77      3979
   macro avg       0.63      0.70      0.65      3979
weighted avg       0.84      0.77      0.79      3979
```

Pros and Cons

Advantages:

- The algorithm is simple and easy to implement.
- There's no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search.

Disadvantages:

The algorithm gets significantly slower as the number of variables increase.

Performance

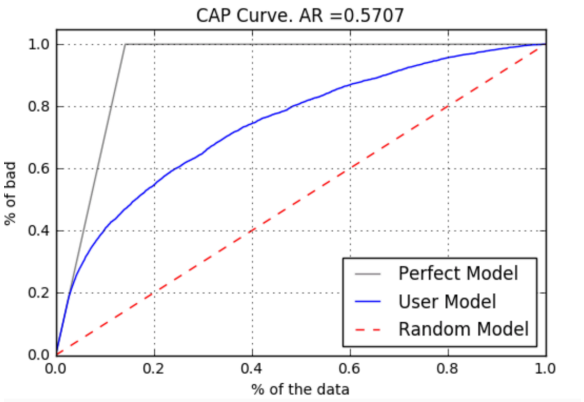
(1) In the Sample

Confusion Matrix:

	0	1
0	24345	685
1	229	483

In the confusion matrix, the true positive is 24345 and the true negative is 483. The false positive is 685 and the false negative is 229. The accuracy is (24345+483)/(35807) = 80.2%. It is the worst performance so far.

CAP curve:



The AR for KNN model is 57.07%. It just outperformed tossing a coin a little bit so this might not be a sufficient model.

(2) Out of the Sample

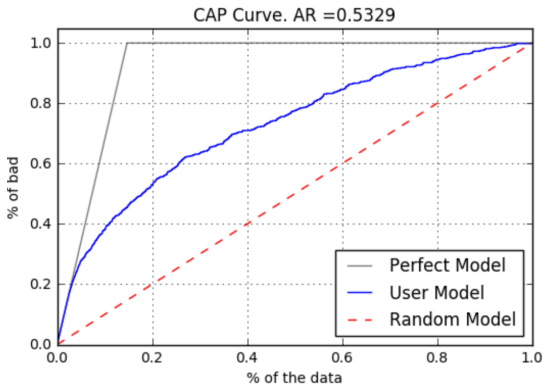
Confusion Matrix:

	0	1
0	2708	685
1	1229	483

In the confusion matrix, the true positive is 2708 and the true negative is 483. The false positive is 685 and the false negative is 1229. The accuracy is (2708+483)/(3979) = 76.8%.

CAP curve:

Measuring Probability of Default Based on Lending Club Historical Data



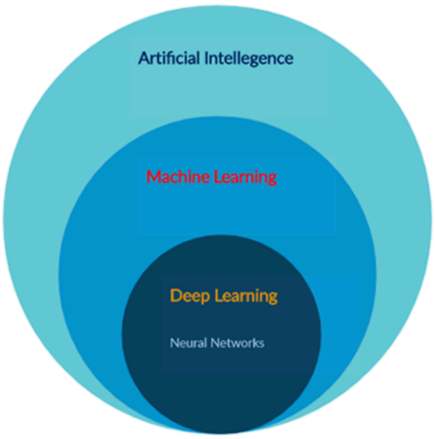
For out of sample test, we got an 53.29% AR which means that it is much worse than the others.

Measuring Probability of Default Based on Lending Club Historical Data

Neural Network

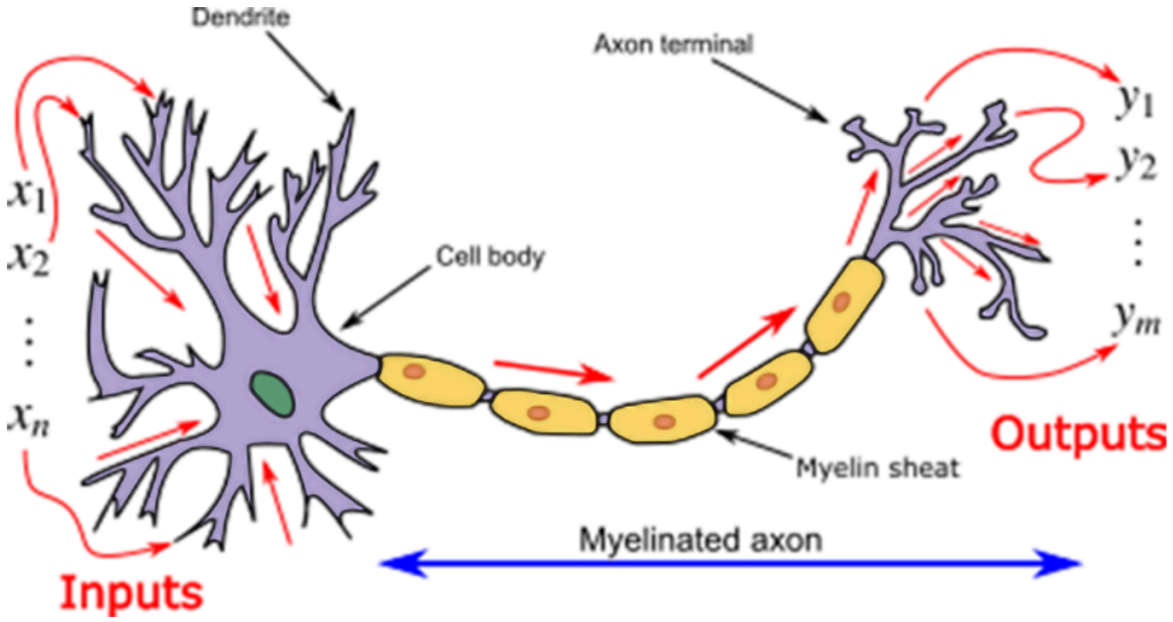
Principle

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks form the base of deep learning, which is a subfield of machine learning, where the structure of the human brain inspires the algorithms.



Neural networks take input data, train themselves to recognize patterns found in the data, and then predict the output for a new set of similar data. Therefore, a neural network can be thought of as the functional unit of deep learning, which mimics the behavior of the human brain to solve complex data-driven problems.

Here is an image representing a biological neuron:

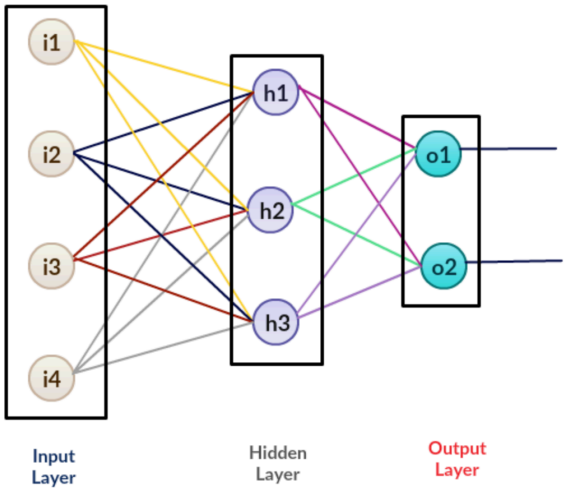


In machine learning, the neurons' dendrites refer to as input, and the nucleus process the data and forward the calculated output through the axon. In a biological neural network, the width of dendrites defines the weight associated with it.

Artificial Neural Networks

In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Measuring Probability of Default Based on Lending Club Historical Data



Process to implement ANN

1. Take inputs.
2. Add bias (if required).
3. Assign random weights to input features.
4. Run the code for training.
5. Find the error in prediction.
6. Update the weight by gradient descent algorithm.
7. Repeat the training phase with updated weights.
8. Make predictions.

Model Specification

There are different varieties of neural networks such as Back Propagation, Convolutional Neural Network and Long short-term Memory Network. However, the relatively simple and popular one is the

Multilayer Perceptron (MLP) which is also called artificial neural network. Normally, it consists of input layer, hidden layer and output layer. We usually called it deep learning if there are three and more hidden layers. Deep learning is the process of training to deep networks.

When the number of hidden layers is relatively low, we normally use gradient descent algorithm like BP. In BP, we need to consider parameters such as learning rate, momentum and number of iterations, etc. However, in MLP algorithm, the most important hyperparameters are hidden layer sizes and maximum iteration.

Hyperparameters for MLPClassifier:

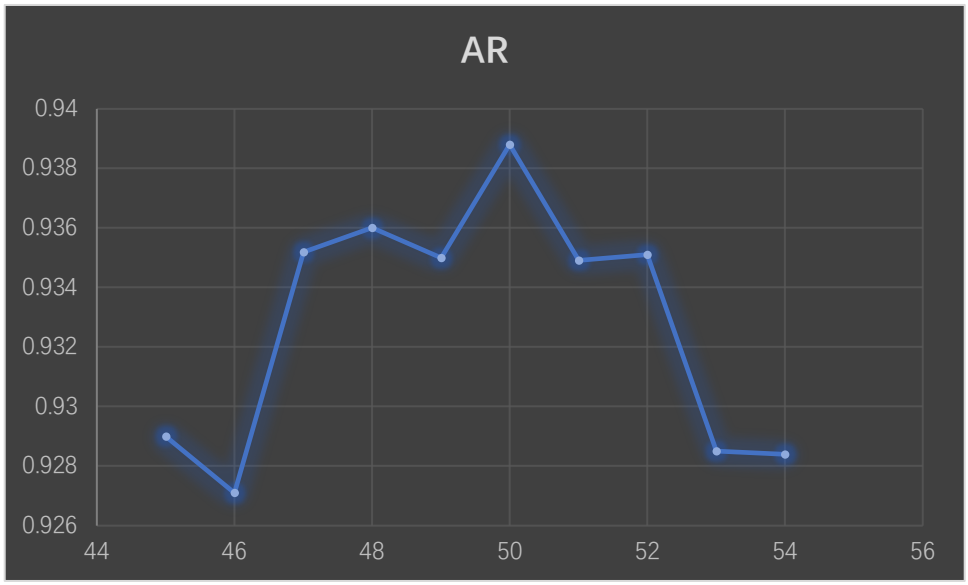
hyper parameter	explanation	input
hidden_layer_size	represents number of neurons in hidden years	50
activation	activation function for hidden layer	default
solver	solver for weight optimizer	default
alpha	regularization term	default
batch_size	size if minibathces for stochastic optimizers	default
learning_rate	learning rate schedule for weight updates	default
learning_rate_init	initial learning rate	default
power_t	exponent for inverse scaling learning rate	default
max_iter	maximum number of iterations	1000

shuffle	whether to shuffle samples in each iteration	default
random_state	determines random number generation	default
tol	tolerance for the optimization	default
verbose	whether to print progress messages to stdout	default
warm_start	check if previous call fits initialization	default
momentum	momentum for gradient update	default
early_stopping	whether to use early stopping	default
validation_fraction	proportion of training data	default
beta_1	exponential decay rate for 1^{st} moment vector	default
bata_2	exponential decay rate for 2^{nd} moment vector	default
epsilon	value for numerical stability	default
n_iter_no_change	maximum number of epochs to not meet tol improvement	default
max_fun	maximum number of loss function	default
nesterove_momentmum	whether to use neterov's momentum	default

Hidden layer sizes and number of maximum iterations deicide the accuracy and training time. We spend most time on optimizing these two parameters. First, for the maximum iteration, we tried the number from default 200 but there will be a warning which illustrates that 200 iterations is not sufficient to get converged result. We keep adding this number to 3000 and we found that it is able to converge and then

we begin to decrease in order to control the time. We finally chose 1000 which wonderfully fits our requirement.

Next, we tried to optimize the hidden layers sizes. Higher hidden layers sizes do not mean higher accuracy because of overfitting. Again, we select some random numbers from 20 to 300 then discovered that the model has better AR around 50. Therefore, we plug in 10 numbers (45 to 54) and figure out that 50 is the optimized one.



Model Performance

(1) In the sample

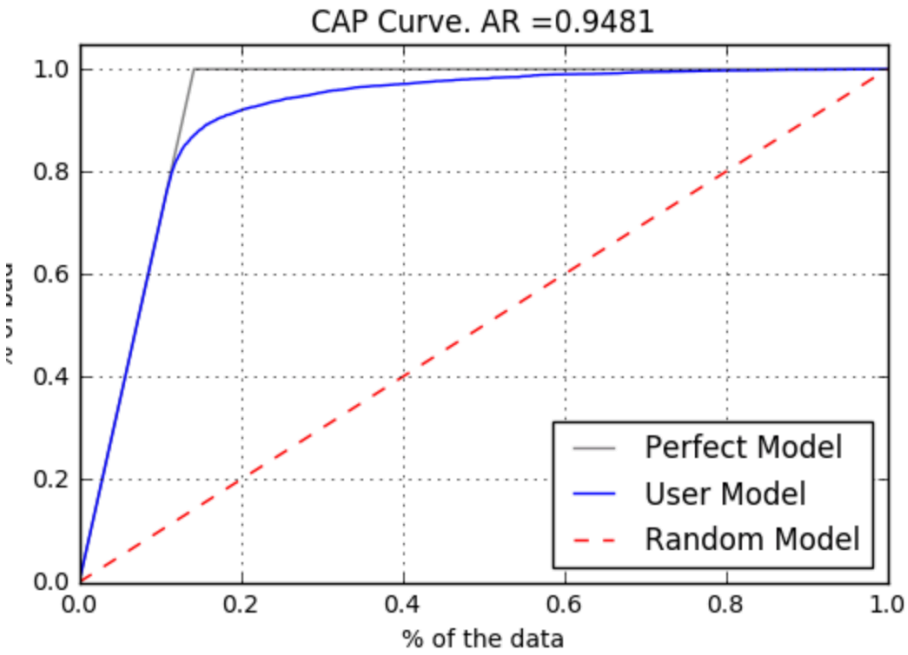
Confusion Matrix:

	0	1
0	30105	618

1	668	4416

In the confusion matrix, the true positive is 30105 and the true negative is 4416. The false positive is 618 and the false negative is 668. The accuracy is (30105+4416)/(35807) = 96.43% which is a great performance.

CAP curve:



In cap curve, the AR for train model is 94.81%.

(2) Out of Sample

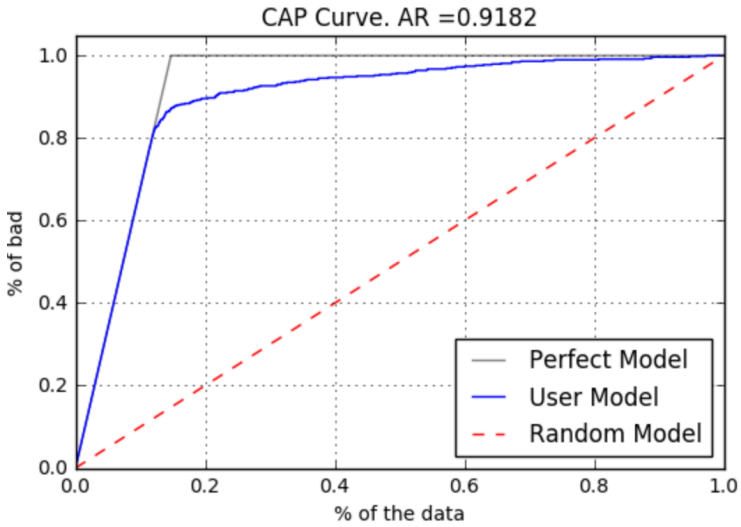
Confusion Matrix

	0	1

0	3330	63
1	79	507

In the confusion matrix, the true positive is 3330 and the true negative is 417. The false positive is 63 and the false negative is 507. The accuracy is (3330+507)/(3979) = 96.41% which is also a great performance.

CAP curve:



In cap curve, the AR for train model is 91.82%.

Pros and Cons

Advantages:

- Capability to learn non-linear models
- Capability to learn models in real-time (on-line learning)

Disadvantages:

- MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.
- MLP is sensitive to feature scaling.

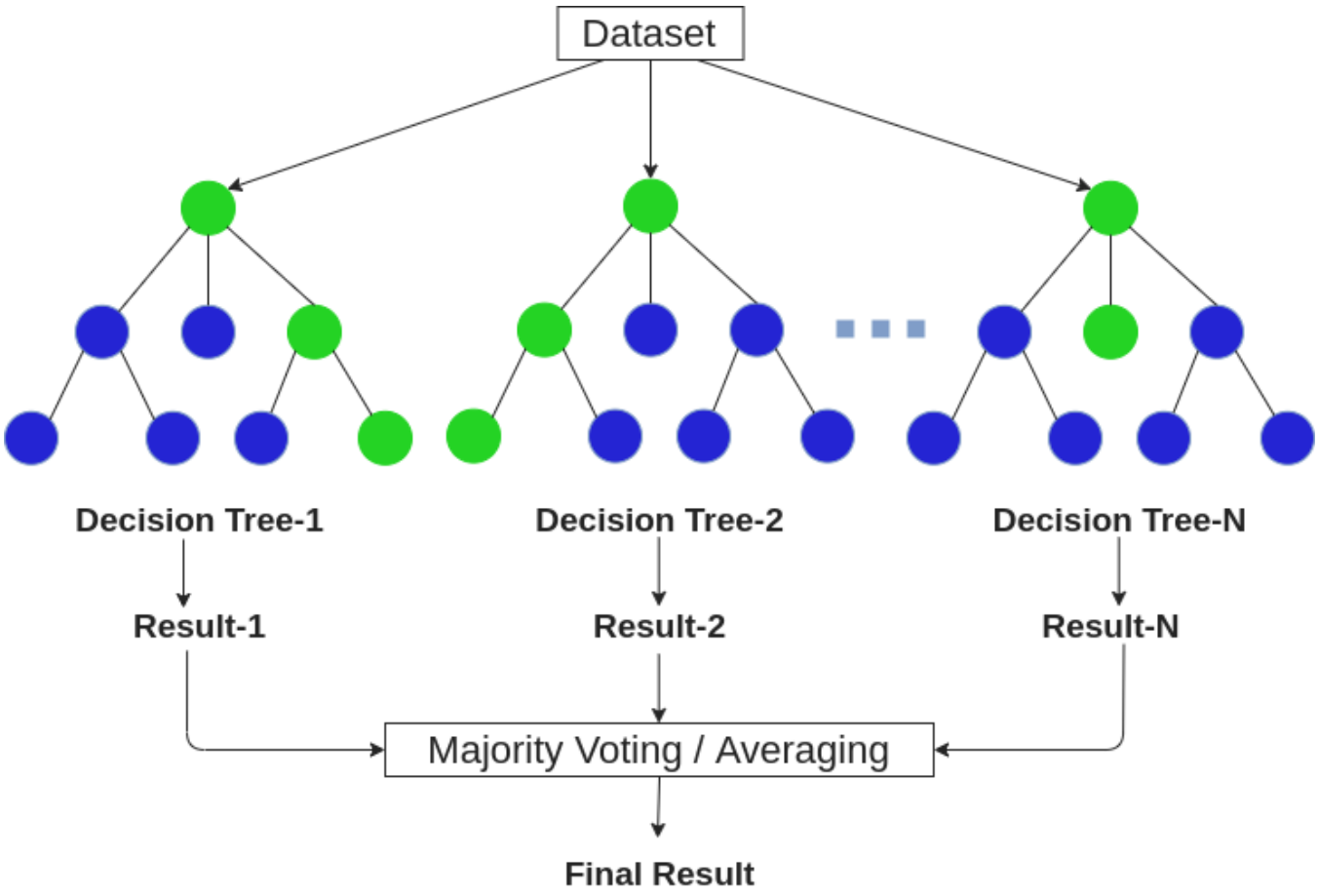
Random Forest

Principle

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled in dependently and with the same distribution for all trees in the forest (Breiman, 2001). Random forests have higher accuracy and generalization ability due to the fact that it operates Bagging and construct a large amount of decision trees in order to decrease the sampling error sufficiently. It can avoid the overfitting in some level.

Bagging is one kind of ensemble learning which is combining weak classifiers to generate a strong classifier. It extract several subsets from original training sample and train each subsets with decision tree or SVM classifier. After getting results from each weak classifier, it will generate the final result through voting process.

Random forest is a modified version of bagging. Instead of considering all the variables, random forest will select some variables stochastically. It avoid repeatability of subsets and more stable than single decision tree.



Model Specification

We used the RandomForesrClassifier() from sklearn.ensemble which contains 19 hyperparameters. The optimization of parameters includes two parts: optimization of bagging and optimization of decision trees.

For bagging, we will try to optimize n_estimators which refers to the number of decision trees in our model. We set a range between 1 to 101 and step is 10 and higher AUC score will be used as our optimization target.

Firstly, we need to test the AUC of default parameters and we can notice that we already got a great accuracy but we still can optimize it to next level.

In [50]:
```
#optimization of bagging
from sklearn.ensemble import RandomForestClassifier
rf0 = RandomForestClassifier(oob_score=True,random_state=10)
rf0.fit(SMOTE_x_train,SMOTE_y_train)
print(rf0.oob_score_)
print("accuracy:%f"%rf0.oob_score_)
```

```
0.9945128895330779
accuracy:0.994513
```

Code below shows that the best n_estimators for our model is 21 and the accuracy is 99.61%.

Next, we will start to optimize the max_features for each decision trees. We set a range between 1 to 11 and step is 1. AUC score is still our optimized target.

In [53]:
```
param_test1 = {"n_estimators":range(1,101,10)}
gsearch1 = GridSearchCV(estimator=RandomForestClassifier(),param_grid=param_test1,
                        scoring='roc_auc',cv=10)
gsearch1.fit(SMOTE_x_train,SMOTE_y_train)


print(gsearch1.best_params_)
print("best accuracy:%f" % gsearch1.best_score_)
```

```
{'n_estimators': 21}
best accuracy:0.996149
```

Then, we can find that the optimal value of max_features is 2.

In [62]:
```
#decision tree parameter optimize
param_test2 = {"max_features":range(1,11,1)}
gsearch1 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=21,
                        random_state=10),
                        param_grid = param_test2,scoring='roc_auc',cv=10)
gsearch1.fit(SMOTE_x_train,SMOTE_y_train)
print(gsearch1.best_params_)
print('best accuracy:%f' % gsearch1.best_score_)
```

```
{'max_features': 2}
best accuracy:0.996283
```

Hyperparameters for RandomForestClassifier

Hyperparameters	Explanation	Input
n_estimators	number of trees in forest	21
criterion	function to measure quality of split	gini
max_depth	maximum depth of tree	25
min_samples_split	minimum number of samples required to split an internal node	15
min_samples_leaf	minimum number of samples required to be at a lead node	default
min_weight_fraction_leaf	the minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node	default
max_features	number of features to consider when looking for the best split	2
max_leaf_nodes	grow trees with max_leaf_nodes in best-first fashion	default
min_impurity_decrease	a node will be split if this split induces a decrease of the impurity greater than or equal to this value.	default

bootstrap	whether bootstrap samples are used when building trees	True
min_impurity_split	threshold for early stopping in tree growth	default
oob_score	whether to use out-of-bag samples to estimate the generalization accuracy	default
n_jobs	the number of jobs to run in parallel	default
verbose	controls the verbosity when fitting and predicting	default
warm_start	when set to `true`, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest	False
class_weight	weights associated with classes in the form `{class_label: weight}`	balanced_subsample
ccp_alpha	complexity parameter used for minimal cost-complexity pruning	default

max_samples	if bootstrap is true, the number of samples to draw from x to train each base estimator	default

Model Performance

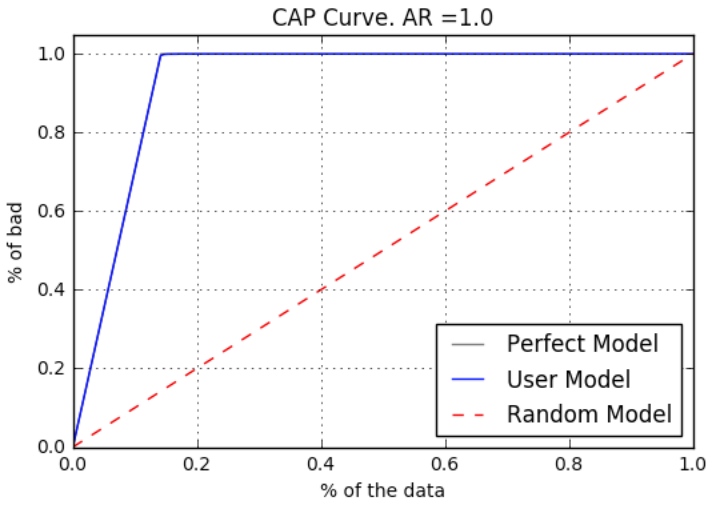
(3) In the Sample

Confusion Matrix:

	0	1
0	30723	0
1	52	5032

In the confusion matrix, the true positive is 30723 and the true negative is 5031. The false positive is 0 and the false negative is 52. The accuracy is (30723+5032)/(35807) = 99.85% which is a awesome performance.

CAP curve:



We can see that we even have a perfect model in CAP curve test which means that our model is sufficient in the sample.

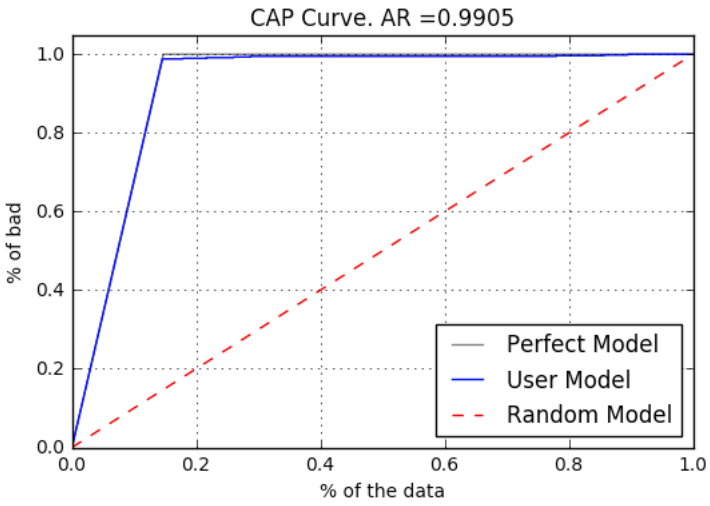
(4) Out of the Sample

Confusion Matrix:

	0	1
0	3393	0
1	7	579

In the confusion matrix, the true positive is 3393 and the true negative is 579. The false positive is 0 and the false negative is 7. The accuracy is (3393+579)/(3979) = 99.82% which is also an amazing performance.

CAP curve:



For out of sample test, we still got an 99.05% AR which is an excellent performance.

Pros and Cons

Advantages:

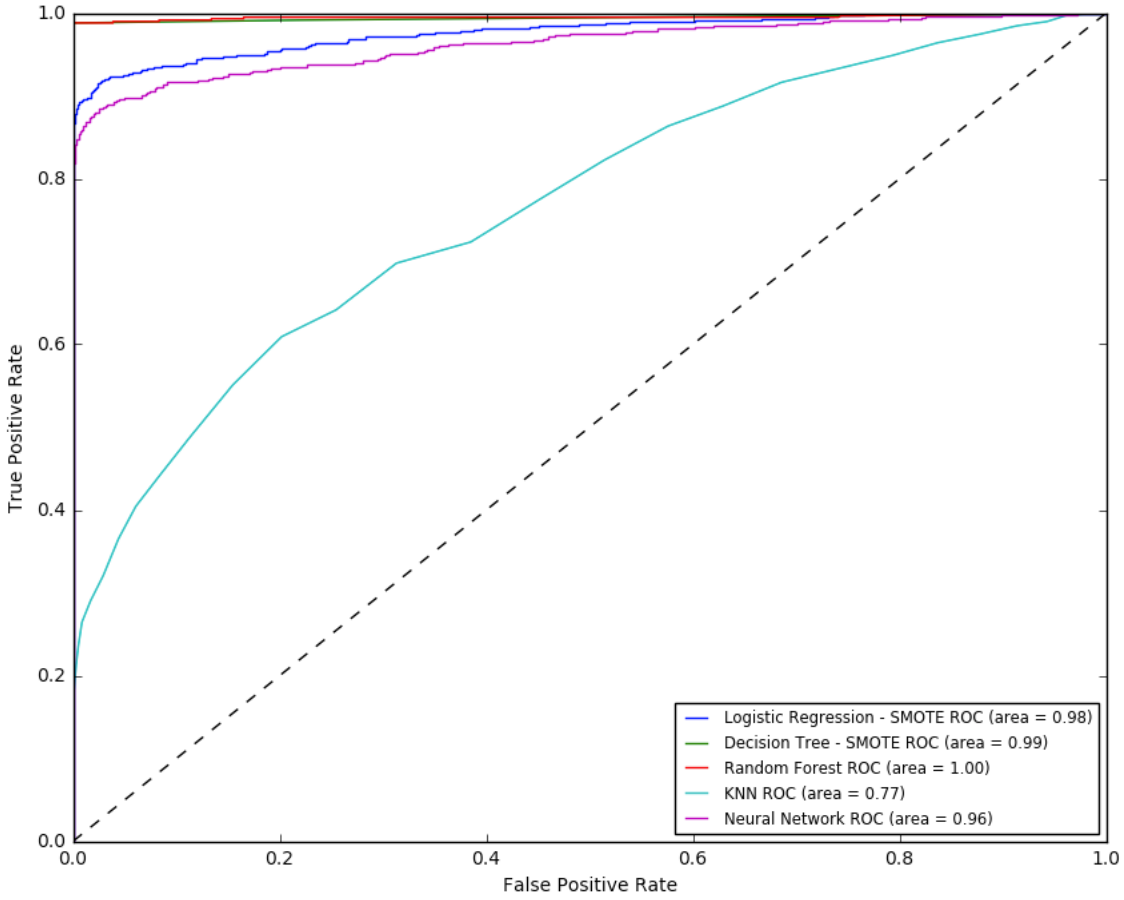
- Powerful to handle large data sets with higher dimensionality
- Be able to balance error in data sets where classes are imbalanced
- Remain high accuracy although missing many features

Disadvantages:

- Tend to overfitting if there is much noise
- Sometimes fell like a black box approach for modelers and we barely have control on it

Conclusion

Totally, we used five models to predict the probability of default but each of them has its own advantages and disadvantages. We believe that there is no perfect model but always better ones. Among these five models, random forest's performance is the best. We plot all the roc curve in one graph to directly compare. The area under the ROC curve (AUC score) will represent the accuracy of each model. The classifier which has better AUC score will have higher accuracy rate.



We can see that KNN classifier has the worst performance because it uses limited nearest sample instead of analyzing class field to classify. For more overlapped data sample, KNN will be a better method comparing with other models.

Random forest classifier has the best performance no matter in confusion matrix, CAP curve and ROC curve. It is simple to use, has great training speed, does well in mass data and is able to balance error in unbalanced sample. Random forest model can be a sufficient tool to help predict probability of default.

References

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Caldieraro, F., Zhang, J. Z., Cunha, M., & Shulman, J. D. (2018). Strategic Information Transmission in Peer-to-Peer Lending Markets. Journal of Marketing, 82(2), 42–63. https://doi.org/10.1509/jm.16.0113

Johri, P., Verma, J. K., & Paul, S. (2020). Applications of Machine Learning.

Lin, Xuchen, Li, Xiaolong, & Zheng, Zhong. (2017). Evaluating borrower's default risk in peer-to-peer lending: Evidence from a lending platform in China. Applied Economics, 49(35), 3538-3545.

Mild, Andreas, Waitz, Martin, & Wöckl, Jürgen. (2015). How low can you go? — Overcoming the inability of lenders to set proper interest rates on unsecured peer-to-peer lending markets. Journal of Business Research, 68(6), 1291-1305.