Exploratory data analysis of LC's 2018 public data set

PRODUCT ANALYST ASSESSMENT

THE ORIGINAL DATA SCIENTIST
FOR GIT PROJECTS

Contents

EDA of Lending Club's dataset

Introduction

Lending Club (LC) provides investors and borrowers a platform for peer-to-peer lending. Currently, using Lending Club's platform, investors can search and browse the loan listings on Lending Club website and select loans that they want to invest in. For making decision about sponsoring the investment they have information supplied about the borrower, amount of loan, loan grade, and loan purpose.

Question 1: Explore the distribution of loans amounts, does the interest rate increase with the size of the loan?

Distribution of loan amounts

From the probability and cumulative distributions, we can see that the

- Most borrowers prefer loan amounts ranging from $1000 - $25,000 .
- Loan amount ranging from $10,000 to $12,000 were applied for the most (approx. 70,000).
- $30,000 - $40,000 was the least preferred loan amount bracket amongst borrowers.

```
Summary statistics for loan amount

count    495242.000000
mean      16025.020394
std       10138.075023
min        1000.000000
25%        8000.000000
50%       14000.000000
75%       22000.000000
max       40000.000000
```

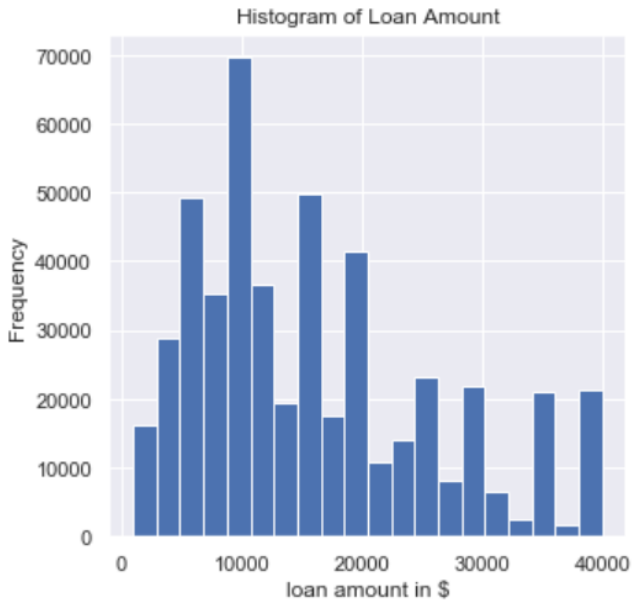


Figure 1. Histogram of loan amount

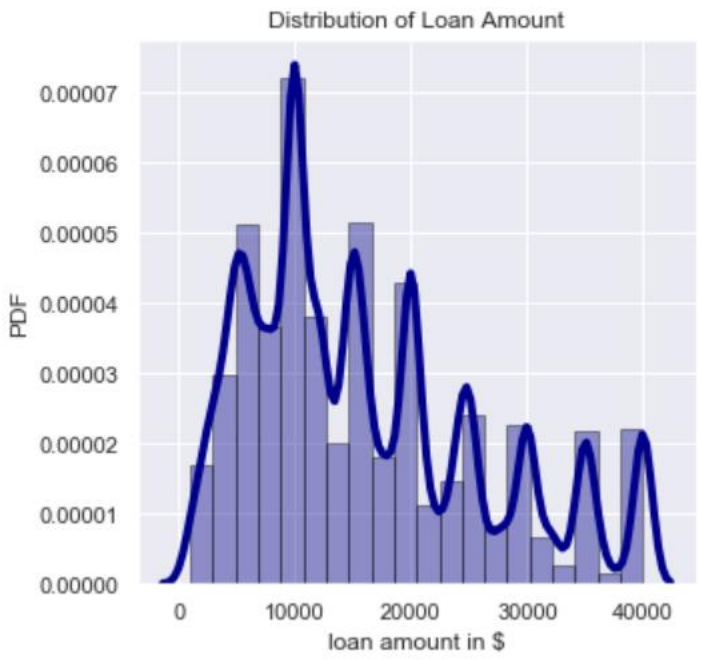


Figure 2. Probability frequency distribution of Loan amount

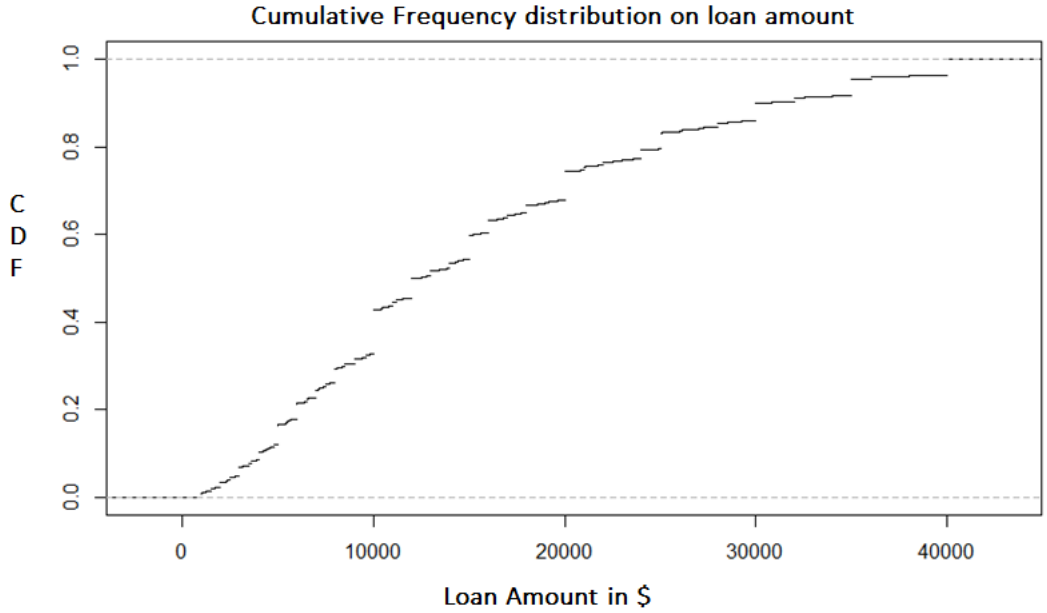


Figure 3. Cumulative distribution function of Loan amount

Dependency of interest rate on loan size

Based on the data dictionary, I have assumed that the loan size is the same as loan amount.

After analysis, I have concluded that interest rate has some relationship with loan amount but loan amount by itself cannot be used predict interest rate. Hence I cannot say that if an interest rate will increase just by increasing loan size but I can say that with increasing loan amount, the most sanctioned interest rate is between 6% - 16%.

Analysis:

1. First step to determine the dependency of two variables is to visualize it. When I plotted interest rate vs loan amount in form of a scatter plot, I didn't see any clear relationship. Although all interest rates are charged for all loan amounts, the most I can say is that at higher loan amounts lower interest rates is preferred over higher interest rates.

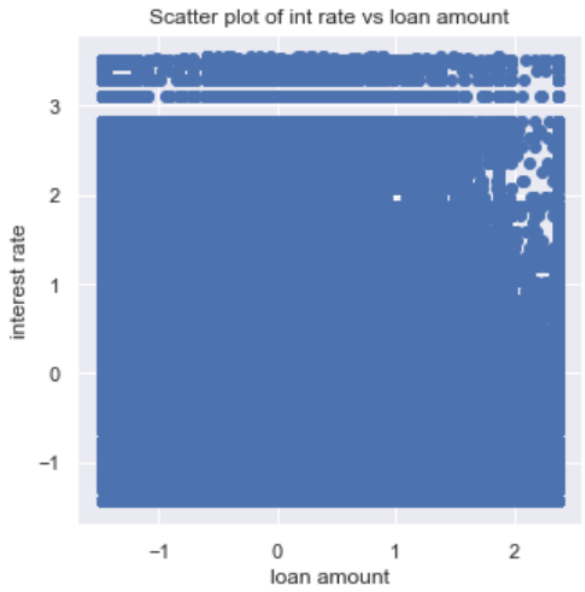


Figure 4. Scatter plot of interest rate vs loan amount

2. The next step in finding dependency, is to see if I can predict interest rate using loan amount.

```
                            OLS Regression Results
==============================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.000
Model:                            OLS   Adj. R-squared (uncentered):              0.000
Method:                 Least Squares   F-statistic:                              7.921
Date:                Mon, 10 Aug 2020   Prob (F-statistic):                     0.00489
Time:                        00:00:48   Log-Likelihood:                     -7.0271e+05
No. Observations:              495242   AIC:                                  1.405e+06
Df Residuals:                  495241   BIC:                                  1.405e+06
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.0040      0.001      2.814      0.005       0.001       0.007
==============================================================================
Omnibus:                    45746.529   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            59420.225
Skew:                           0.826   Prob(JB):                         0.00
Kurtosis:                       3.387   Cond. No.                         1.00
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

intercept: -3.1977420567489023e-16
slope: [0.0039992]
coefficient of determination: 1.6e-05
```

Figure 5. Regression model parameters

R-squared value and p-value of the linear regression model is very low. Low R-squared value indicates that the loan amount cannot accurately predict the value of interest rate. When R-squared is low, low P values still indicate a real relationship between the independent variable and the dependent variable, it just cannot be used for prediction.

3. Step 2 shows that there could be some kind of relationship between loan amount and interest rate. I tried to explore the relationship further.

   I created classes for interest rate called 'Rate class' with a step size of 2% and plot this against loan amount.

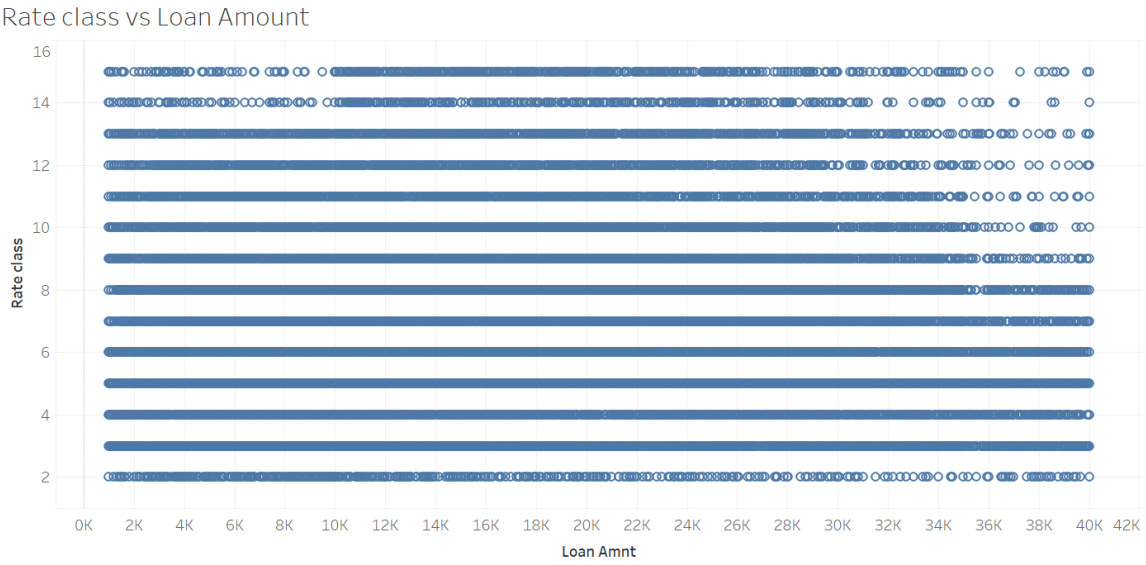
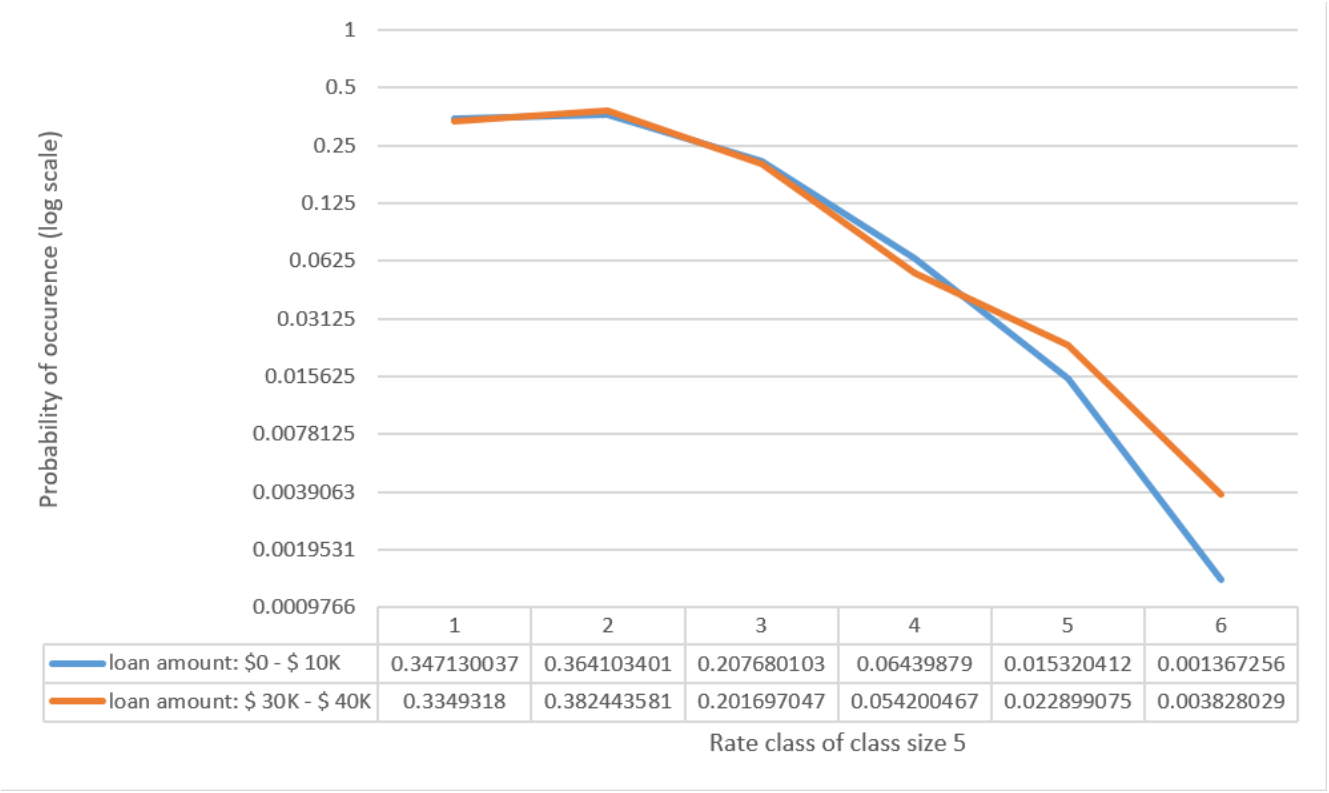


Figure 6. Scatter plot of rate class vs loan amount

It is clear from Figure 6 -

- Higher interest rates (28 % - 30 %) are generally not sanctioned for many loan amount. But it is especially rarely sanctioned for amounts
  - less than $ 10,000
  - greater than $ 28,000
- Lower interest rates (0% – 4%) is less often sanctioned for higher loan amounts
- With increasing loan amount the most sanctioned interest rate is between 6% - 16%



	1	2	3	4	5	6
loan amount: $0 - $ 10K	0.347130037	0.364103401	0.207680103	0.06439879	0.015320412	0.001367256
loan amount: $ 30K - $ 40K	0.3349318	0.382443581	0.201697047	0.054200467	0.022899075	0.003828029

Figure 7. Probability of occurrence of a rate class with different loan class.

It is clear from Figure 7 -

- When comparing two groups of loan amounts, loan amount ranging between $30,000 to $ 40,000 have higher probability of high interest rates sanctioned as compared to loan amount ranging between $10,000 to $ 20,000

Question 2: Do the monthly payment (installment) and interest rate together have any effect on the loan status?

Yes, installments and interest rate together have an effect on loan status.

To measure the effect of installments and interest rates on loan status, I tried to see if I can predict loan status using these two features. I used logistic regression model to determine if I can classify a loan status into delinquent and non-delinquent based on the values of installments and interest rate. I have assumed that any loan with status 'Current' and 'Paid-off' will be non-delinquent and loan with status 'charged-off', 'Default', 'In grace period', 'Late (16 – 30 days)' and 'Late (31 – 120 days)' will be delinquent.

The resulting model had an f1 score of 61% and accuracy of 62%. Although f1-score and accuracy is closer to 50%, it still indicates that **the model is able to predict the loan_status based on interest rate and installment up to certain confidence.**

	precision	recall	f1-score	support
0	0.61	0.66	0.64	5297
1	0.63	0.58	0.61	5322
accuracy			0.62	10619
macro avg	0.62	0.62	0.62	10619
weighted avg	0.62	0.62	0.62	10619

Figure 8. Model Evaluation

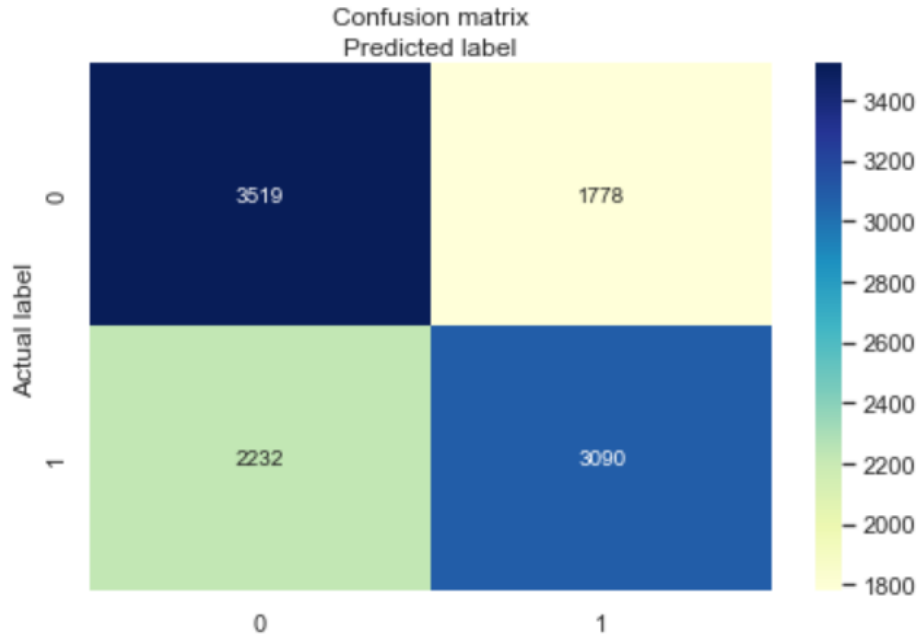


Figure 9. Confusion Matrix

Biggest challenge faced in creating the model was the highly imbalanced dataset. Data points are more skewed towards non-delinquent (loan status of Current and fully paid) class. To improve the accuracy and f1-score of the model, we should include more data points related to delinquent class (loan status other than current and fully paid). To tackle class imbalance, I employed first under-sampling of majority class and then over-sampling of the minority class.

Question 3: Any loan that does not have a loan_status of current or paid_off is considered a delinquent loan. How would you go about predicting which loans will go delinquent? You should not build a model but instead breakdown an approach with a detailed plan.

I would use a Neural Network to predict which loans will go delinquent.

Steps for building Neural Network prediction model:

Pre-processing Data

1. Analyze the data for missing values and corrupt data. To handle bad data, we can
    a. remove rows with null or corrupt value in all columns

In LC's dataset, to pre-process the data I have

a. dropped the columns 'id','member_id','url' and 'desc' because it has all NULL/NA values
b. dropped rows that had NULL values in all columns
c. In columns with numerical values, replaced null values with (maximum value + 1). This will create its own label
d. Split one date field into 3 numerical fields one for year, one for months and one for day.
e. Extract only first 3 values of zip-code field

I am going to keep the columns that have many NULL values (ex – deferral_term has only 277 non-null values out of 495k rows) and let the neural network model decide its significance in predicting the loan status.

2. Change the categorical columns (text or Boolean) into numerical categories. In case the categorical column has a blank value, group all blanks into one category and assign it a label of its own.

   In LC's dataset, I would change columns like grade, sub-grade, home_ownership etc into numerical categories.
   Ex – Column home_ownership has values 'Any', 'Mortgage', 'Own' and 'Rent'. I would turn these categories into numerical categories. Although the dataset doesn't have a blank value in home_ownership column, but if it did I would assign it a number of its own.

Values	Numerical Category
Any	1
Mortgage	2
Own	3
Rent	4
Blank values	5

3. Once all features have been converted into numerical data, go ahead and normalize all features.
   The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

   I have normalized the features in LC's dataset because they have different scales.

4. Split the dataset into Train and Test data with 80 : 20 split ratio

5. If the dataset is imbalanced then balance the Train set using under-sampling and/ or over-sampling techniques.

   In LC's dataset is heavily biased –

Loan status	Count of records
Current	437,318

Fully Paid	40,240
Delinquent	17,684

I would use both under-sampling and over-sampling techniques, I would plan to under-sample current and oversample Fully-paid and Delinquent loan status category.

Loan status	Count of records (Train)	Sampling technique
Current	349,854	Under-sample to 1/4th its original size
Fully Paid	32,192	No change
Delinquent	14,147	Oversample to twice its original size

Feature Selection

I would select all features resulting from data pre-processing step to be used as input values in the neural network and let the model train itself based on all features to effectively predict 'loan status'.

Neural Network structure

A Neural Network consists of the following components

- An input layer, *x*
- An arbitrary amount of hidden layers
- An output layer, $\hat{y}$
- A set of weights and biases between each layer, *W and b*
- A choice of activation function for each hidden layer, σ. Ex – Sigmoid or RELU.
- Classifier Function Ex – Softmax or SVM

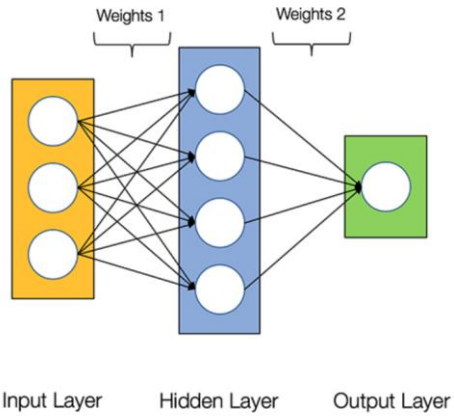


Figure 10. Architecture of a 2-layer neural network

In our Neural Network,

1. The input layer will contain all features except 'loan_status'. That gives us 140 features. So the input layer will have 140 nodes. Mathematically it will be a matrix with 140 rows and 1 column.

2. Output layer will contain two output nodes one for Non-delinquent (Loan status = 'current' and loan status = 'paid-off') and other for Delinquent (Loan status != 'current' or loan status != 'paid-off'). I chose two nodes

because I will use a classifier to split the output in 2 groups. Mathematically it will be a matrix with 2 rows and one column. I will assume that node 1 is for non-delinquent and node 2 is delinquent.

3. Since Neural Network is a heuristic approach, I will start with 1 hidden layer and it will contain 71 neurons (mean of nodes in input and output layers). I will adjust the number of hidden layers or the number of neurons in the hidden layer the after one training is complete and if the result is not satisfactory.

4. Weights and Biases
   Weights and biases are essentially matrices at each hidden layer in the neural network which helps to transform input features and map it to the output nodes. So for determining weights and biases for our model, I will first decide the dimension of matrices and initialize the matrices.

   The number of weights used will depend on the number of hidden layers in the network. In my network, I have one hidden layer so I will need two weight matrices. First weight (W1) in combination with bias (B) will be used to map input features to the neurons of hidden layer. The second weight (W2) will map the neuron values to output nodes.

   The dimension of W1 weight matrix will depend on the number of input nodes and if it is fed into the model as columns or rows, and the number of neurons in the hidden layer. In our model, input layer has 140 features arranged in a matrix of dimension - 140 rows X 1 column, so the weight matrix has 140 columns. The number of neurons in the hidden layer is 71, so the weight matrix will have 71 rows. I will initialize the weight matrix with values on a Gaussian distribution with mean 0 and standard deviation 1.

   The B bias matrix is a collection of constants that will be added to the dot product of weight and input value to provide extra tuning capabilities. Therefor the bias matrix will have the one column and number of rows equal to the number of neurons. In our model the bias matrix will have one column and 71 rows. I will initialize the weight matrix with values on a Gaussian distribution with mean 0 and standard deviation 1.

   The dimension of W2 weight matrix will depend on the number of output nodes and the neurons in the hidden layer. In our model I have two output nodes, so the number of rows will be 2 and there are 71 neurons in the hidden layer so the number of columns will be 71. I will initialize the weight matrix with values on a Gaussian distribution with mean 0 and standard deviation 1.

   Therefore,

   Weight matrix = W1 = matrix with 71 rows X 140 columns = dimension [71 x 140]
   having values w_{ij} where w_{ij} is a value on Gaussian distribution (mean = 0, std = 1) and i is the number of rows and j is the number of columns

   Bias matrix = B = matrix with 71 rows and 1 coumn = dimension [71 x 1]
   having values b_i where bi is a value on Gaussian distribution (mean = 0, std = 1) and i is the number of rows

   Weight matrix = W2 = matrix with 2 rows X 71 columns = dimension [2 x 71]
   having values w_{ij} where w_{ij} is a value on Gaussian distribution (mean = 0, std = 1) and i is the number of rows and j is the number of columns

5. Activation function is used to add non-linearity to the model. This helps specifically when we want to use classification. For our model, I will use sigmoid function. Sigmoid function is good non-linear function to use if case the output layer has binary nodes.

   The Sigmoid is useful as an activation function because inputs way greater than 0 will quickly result in a number very close to 1, once passed through the sigmoid function. By the same principle, the smaller the input, the closer the resulting number will be to 0. Input numbers very close to 0 will have a value that smoothly oscillates between 0 and 1.

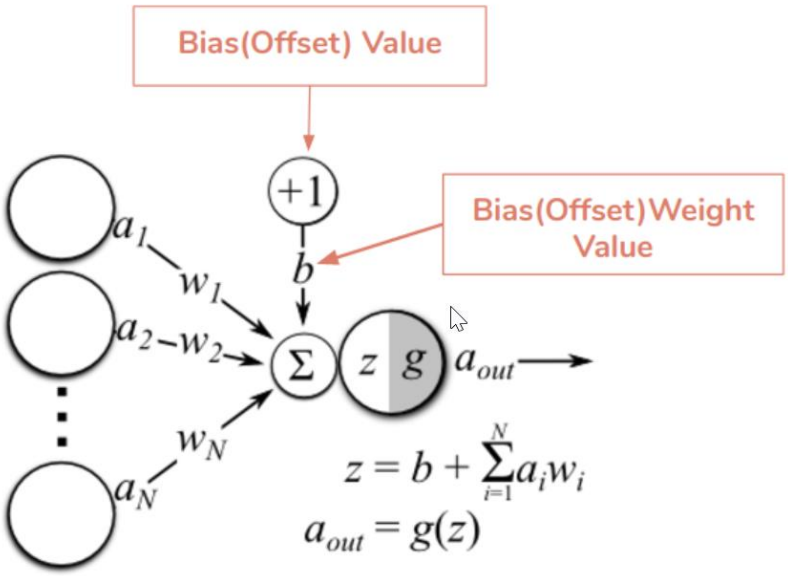


Figure 11. Representation of how (weights + biases) and activation function is applied in transforming features to a neuron. a_{out} denotes a neuron of a hidden layer and becomes input to the neuron of next hidden layer or W2. Source hackernoon.com

6. Using Weights and Activation functions, we transform and map the input feature to output nodes. Once the features are mapped to output nodes we use a classifier to determine the weight / probability of each output node (this will be the predicted value) and how far apart is the predicted node value from the actual value.

   In our model, I will use Softmax classifier for this purpose.

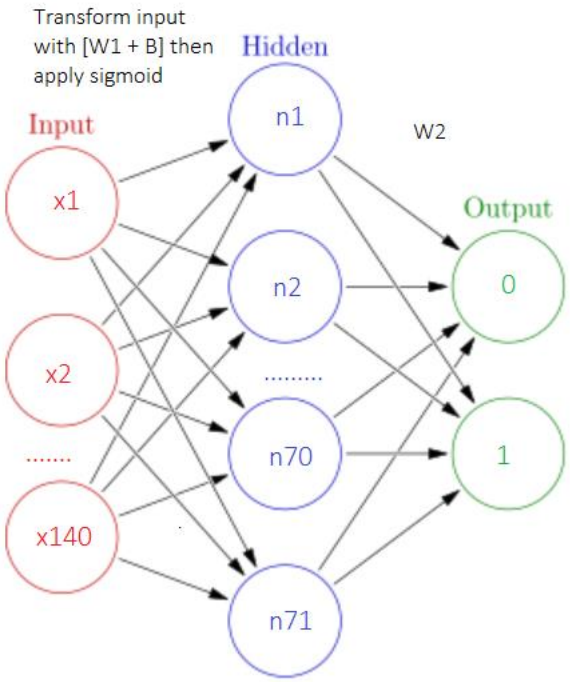


Figure 12. Neural Network Structure

7. I will start with a learning rate of 0.01 and after every 40,000 rows of the Train dataset, I will reduce the learning rate by a factor of 10.

8. I will use a cost function to predict the difference between predicted value and actual value. Cost function basically tells us how bad the predictions are. I will use Mean Squared error (MSE) as the cost function in this model. I will subtract the expected output value from the predicted output values and square the result for each neuron. Summing up all these squared errors will give me the final value of our cost function.

Training Model

I will use the following terms in describing the process of training the model

- A train dataset is a table with 141 columns (140 features + loan status) and approximately 147,950 rows (after balancing the train dataset)

- Forward propagation: Forward propagation means we are moving in only one direction, from input to the output, in a neural network

- Backward propagation: Back-propagation is just a way of propagating the total loss back into the neural network to know how much of the loss every node is responsible for, and subsequently updating the weights in such a way that minimizes the loss by giving the nodes with higher error rates lower weights and vice versa.

- One Training iteration consists of
    - passing one row of data with 140 features to the model
    - forward propagating the input variable to the output nodes
    - predicting the output node
    - finding the difference between the actual and the predicted value

- Back propagating the difference to train the weights

I have assumed that the first node or first row of output matrix will be non-delinquent and second node or second row of output matrix will be delinquent.

I will use the following steps to train the model -

1. Take one row of the Train dataset. 140 features

2. Transform the data set into a matrix with 140 rows and 1 column. Let's call this matrix X with dimension [140 x1]

3. Create an expected value matrix called E with dimensions [2 x 1]. If the row passed to the model corresponds to delinquent loan status, the value to matrix will be: row 1 = 0 and row 2 = 1. If the row passed to the model corresponds to non-delinquent loan status, the value of expected matrix will be: row 1 = 1 and row 2 = 0.

4. Set the learning rate for the iteration. Let's call this alpha. I will begin with alpha = 0.01

5. Forward propagate the input variables through the model
   3.1 Take the dot product of W1 weight matrix [71 rows X 140 columns] with input matrix [140 rows X 1 columns].
   Mathematically this is W.X resulting in matrix of dimension [71 x 1]
   3.2 Add the bias matrix to the matrix obtained in step 3.1
   Mathematically this is W.X + B resulting in matrix of dimension [71 x 1]
   3.3 Apply sigmoid function to the matrix obtained in step 3.2. The result will be a matrix with 71 rows and 1 column. Let's call it H1 matrix with dimension [71 x 1]
   Mathematically it is applying the formula $(1 / 1 + e^{-x})$ on each element of the matrix obtained in step 3.2 (x = matrix element). This matrix represents the 71 neurons of the hidden layer.
   3.4 Now I need to transform and map the neurons of hidden layer to the output. For this take the dot product of Weight W2 [2 x 71] and neuron [71 x 1] matrix. This will result into output matrix O1 with a dimension of [2 x 1].

6. Apply the Softmax classifier function on the output matrix to find the probability of the output nodes. The resulting matrix O with dimensions [2 X 1] will have the probability of occurrence of the output nodes. Row 1 will denote the probability of occurrence of Non-delinquent and Row 2 will denote the probability of occurrence of delinquent.

7. Next compare the output matrix O with the expected matrix E. If the model made correct predictions then the probability of predicted loan status would be 1 and the other would be 0. In most situation this won't be the case. So subtract the values in output matrix O from the values in expected matrix E. This will give us a delta matrix. Let's call this matrix D

8. Next calculate the cost of the iteration using the MSE formula. For the example below the cost of iteration will be 0.004 (for row 1) + 0.001 (for row 2) = 0.005 .

```
OUTPUT LAYER:       EXPECTED VALUES:        COST:
[                   [                       [
    0.8,                1,                      (0.8 - 1) ** 2,

    0.1,                0,                      (0.1 - 0) ** 2,
```

Figure 13. Calculation of cost.

The idea here is to tweak the weights and biases of each layer to minimize the cost function. The smaller the value of the MSE, the closer will the predictions be to the actual results.

9. Back propagate the delta matrix to adjust the weights and biases
    - 9.1 Apply Softmax*(1 – Softmax) on each element of D matrix to back-process Softmax. This will create a new matrix, let's call this delta-dash or D' [2 rows x 1 column]
    - 9.2 For finding delta-W2 matrix, take dot product of D' [2 x 1] with transpose of H1 [1 x 71] matrix
      Mathematically, it is delta-W2 = $D'.H^T$ with dimension [2x71]
    - 9.3 Add W2 and (delta-W2)*alpha to get adjusted W2 matrix
    - 9.4 For finding delta-H1 matrix, , take dot product of transposed W2 matrix [71 x 2] with D' [2 x 1]
      Mathematically, it is delta-H1 = $W2^T.D'$ with dimension [71x1]
    - 9.5 Delta-H1 is back propagated further to back-process sigmoid function. Apply Sigmoid*(1-Sigmoid) on each element of delta-H1. This will create a new matrix [71 x 1] called delta-H1'
    - 9.6 Adjust B matrix by adding (delta-H1')*alpha
    - 9.7 For finding delta-W1 matrix, take dot product of delta-H1' [71 x 1] with transposed input matrix X [1 x 140] matrix.
      Mathematically, it is delta-W1 = delta-$H1'.X^T$ with dimension [71 x 140]
    - 9.8 Add W1 with (delta-W1)*alpha to adjust W1 weight matrix.

This finishes one iteration of training the model. The adjusted weights W1, W2 and bias B will now be used for the next iteration of training.

Training is considered complete when the model has run through each row in the Train dataset and the accuracy reaches an expected value.

If the accuracy is below what we expected, I will play around with adjusting one or more of the below and retrain the model.

- learning rate (Increase the learning rate of minority class to overcome imbalance data)
- number of hidden layers
- dimension of weight matrix
- activation function

Test the model

After every 1000 iterations of Train dataset, Run the model on the test dataset. The model will run once for each row in the dataset. This will complete one round of testing.

Pass one row of test dataset through the model and calculate the cost and accuracy similar to how we train the model. (steps 1 through 8). The important thing to note is that there is no back propagation in Testing. We stop at finding the delta and calculating the cost.

For each test iteration, calculate accuracy and cost and plot it. When analyzing the plot after one round on testing, the accuracy should increase and cost should decrease.

For calculating the overall accuracy and overall cost of a testing round, take the mean of all accuracy and mean of all costs of each test iteration within a test round. Then plot the overall cost and accuracy of all test rounds. The x-axis of the plot will be the number of test round and y –axis will be overall accuracy and cost.

As the model completes more and more testing rounds, the accuracy should increase and the cost should go down. If this isn't happening then I will play around with adjusting one or more of the below and train the model again from scratch.

- learning rate
- number of hidden layers
- dimension of weight matrix
- activation function

Model Tuning

After Training the model once, we can check the weight and bias matrix for sparsity. If we find any, we can apply optimization techniques like pruning.

We will also analyze the weight matrix to see which feature has more weight attached. Features with higher weights means it will impact the model more, so we can collect similar data points. For example, if we see that the personal information of the borrower has more impact on the model, then we can collect more information about the borrower to improve the model.

Conclusion

Finally when the model is completely trained, it will be able to predict the loan status accurately and we could use this algorithm to improve LC's decision of accepting a loan on their site. This in turn will improve the quality of loans that LC puts out on its platform for peer to peer lending and improve customer (investor) satisfaction.

Code:

```
###--- Importing required packages ---###
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
import statsmodels.api as sm
import scipy
import random

# for inline plots in jupyter
%matplotlib inline

# for logistic regression
from sklearn.linear_model import LogisticRegression

# for model evaluation
from sklearn.metrics import confusion_matrix, classification_report

# for pre-processing
from sklearn import preprocessing

# for fitting distribution
from scipy import stats

# for pipeline and SMOTE
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline
from imblearn.pipeline import pipeline

from sklearn.model_selection import GridSearchCV


###--- Data Processing ---###
# Reading Data
df1 = pd.read_csv("loans_2018.csv")
df1.head()

# getting metrics on data
print(df1.shape)
df1.describe()

# just from looking at the count, I could say that not all rows of the
1048575 have values in all the columns
# Also the columns id and member id have only null values, so we can get rid
of these two

# Data wrangling
df2 = df1[df1.columns[2:145]]
```

```
df3 = df2.dropna(axis=0, how = 'all')
print(df3.shape)
df3.head()

#----- Question 1: Explore the distribution of loans amounts, does the
interest rate increase with the size of the loan? -----#

# settings for seaborn plotting style
sns.set(color_codes=True)

# settings for seaborn plot sizes
sns.set(rc={'figure.figsize':(5,5)})

##-- distribution of loan amount --##

#- Density Plot and Histogram of all loan amount -#
x0 = df3['loan_amnt']

print("Summary statistics for loan amount")
print("    ")
print(df3['loan_amnt'].describe())

plt.hist(x0, bins = 20)
plt.title('Histogram of Loan Amount')
plt.xlabel('loan amount in $')
plt.ylabel('Frequency')
plt.show()

sns.distplot(x0,
             hist=True,
             bins=20,
             color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})

plt.title('Distribution of Loan Amount')
plt.xlabel('loan amount in $')
plt.ylabel('PDF')

plt.show()

# checking for gamma distribution fit
# sns.distplot(x0,
#              fit=stats.gamma,
#              kde=False)

# plt.title('Distribution of Loan Amount')
# plt.xlabel('loan amount in $')
# plt.ylabel('PDF')
```

```
# plt.show()

#- cumulative distribution function -#

# Values of the CDF
xx = np.array(df3['loan_amnt'])
cdf = scipy.stats.norm.cdf(xx, 16025, 10138)
plt.rcParams['agg.path.chunksize'] = 10000

# Make the plot
plt.plot(xx, cdf)
plt.xlabel ("loan amount")
plt.ylabel("CDF")
#plt.margins(x=10)

plt.show()

##-- dependency of interest rate on loan amount --##

#- visual representation of the loan amount and int rate -#
x1 = np.array(df3['loan_amnt']).reshape(-1,1) #.reshape() on x because this
array is required to be two-dimensional,
                                              # or to be more precise, to
have one column and as many rows as necessary
y1 = np.array(df3['int_rate'])

x_norm = preprocessing.scale(x1)
y_norm = preprocessing.scale(y1)

print(x_norm)
print(y_norm)
print("   ")

plt.scatter(x_norm, y_norm)
plt.title('Scatter plot of int rate vs loan amount')
plt.xlabel('loan amount')
plt.ylabel('interest rate')
plt.show()

# It is not clear looking at the visualization what the relation between
interest rate and loan amount could be.
# A good way to find the dependency is through linear regression

# #- Linear Regression -#

model = sm.OLS(y_norm, x_norm)
results = model.fit()
print(results.summary())

# #- Linear Regression -#
```

```
from sklearn.linear_model import LinearRegression

# creating model
model = LinearRegression().fit(x_norm,y_norm)

# Analyzing the model

print("   ")
print('intercept:', model.intercept_) #model parameters
print('slope:', model.coef_)

r_square = round(model.score(x_norm, y_norm),7) # coefficient of
determination ($R^2$).
# since p-value and R-squared are both low, one variable cannot be used for
predicting the other but they are correlated.

print('coefficient of determination:', r_square) # in general, the higher
the R-squared, the better the model fits your data.

#----- Question 2: Do the monthly payment (installment) and interest rate
together have any effect on the loan status? -----#

##-- Filtering Data --##

df4 = df3[['int_rate','installment','loan_status']]
print(df4.shape)
df4.head()

# we then classify current and fully-paid as non-delinquent by assigning it
value of 0 and the rest as delinquent by assigning it a value of 1.
# By assigning delinquent a value of 1 we are now going to predict for
delinquent.

df4['loan_class'] = df4['loan_status'].apply(lambda x: 0 if x == 'Fully
Paid' or x == 'Current' else 1)
print(df4.shape)
df4.head()

#-- Undersampling the dataset --#
y = df4['loan_class']
x = y.values.tolist()
t = 0

for a in range(len(x)):
    if (x[a] == 0 and bool(random.getrandbits(1))):
        x[a] = 2
        t = t + 1
    if t == 30648:
        break
```

```
# had to adjust the value to t after I made one round of prediction to tweak
the accuracy and f1-score

df4['column1'] = x
df4 = df4[df4.column1 != 0]

print(df4.shape)
df4.head()

######------ Logistic Regression Modelling ------######

#-- defining predictors and response variables --#
X = df4[['int_rate','installment']] # split features from loan class
Y = df4[['loan_class']]

#-- creating training and test data --#

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30,
random_state = 23, shuffle = True)

#-- verifying the training and test data --#
print(X_train.shape)
print(Y_train.shape)
print("    ")
print(X_test.shape)
print(Y_test.shape)
print("    ")

#-- verifying the train and test data has equal split --#

num_neg_train = (Y_train==0).sum()
num_pos_train = (Y_train==1).sum()

num_neg_test = (Y_test==0).sum()
num_pos_test = (Y_test==1).sum()

print(num_neg_train)
print(num_pos_train)
print(round(num_neg_train/num_pos_train,2))
print("    ")
print(num_neg_test)
print(num_pos_test)
print(round(num_neg_test/num_pos_test,2))

# ## I am commenting the section because I will do undersampling before
splitting in test and train data.
# ## I am going to do this because even the Test data is heavily biased, so
I will under-sample before train - test split
```

```
#########################################################################
# # the dataset is highly imbalanced with the ratio of 0 : 1 = 27, so we
will balance the train dataset by undersampling.

# ##-- dataset balancing by under-sampling --##
# #y = df4['loan_class']
# y = Y_train['loan_class']
# #df4
# x = y.values.tolist()
# t = 0

# for a in range(len(x)):
#     if (x[a] == 0 and bool(random.getrandbits(1))):
#         x[a] = 2
#         t = t + 1
#     if t == 12362:
#         break

# X_train['column1'] = x
# Y_train['column1'] = x

# X_train = X_train[X_train.column1 != 0]
# Y_train = Y_train[Y_train.column1 != 0]

# X_train = X_train[['int_rate','installment']]
# Y_train = Y_train[['loan_class']]

# print(X_train)
# print(Y_train)

#########################################################################

#-- multicollinearity between the data points --#

# scatter plot

x2 = X_train['int_rate']
x1 = X_train['installment']

plt.scatter(x1, x2)
plt.title('Scatter plot of independent variable')
plt.xlabel('int_rate')
plt.ylabel('installment')
plt.show()

# correlation coefficient
corr_df = X_train.corr(method = 'pearson')

sns.heatmap(corr_df, cmap = "YlGnBu", vmax = 1.0, vmin = -1.0, linewidth =
2.5)
```

```
print(corr_df)

plt.yticks(rotation = 0)
plt.xticks(rotation = 90)
plt.show()

# output
#              int_rate  installment
# int_rate     1.000000     0.076046
# installment  0.076046     1.000000


# so there is some correlation between interest rate and installment but it
isn't very significant.
# We can proceed with taking interest rate and installment as the features
of the logistic model

#-- Model using Scikit-learn package--#

# creating model
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression()
logreg.fit(X_train,Y_train)
y_pred_skl=logreg.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_skl))

# confusion matrix accuracy of the model
cnf_matrix = metrics.confusion_matrix(Y_test, y_pred_skl)

print(cnf_matrix)

# create heatmap

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
# output:
#       precision    recall   f1-score   support

#   0         0.61      0.65       0.63      5315
#   1         0.63      0.59       0.61      5304
#
# accuracy                           0.62     10619
# macro avg  0.62      0.62       0.62     10619$
# weighted avg 0.62      0.62      0.62     10619

# confusion matrix    [[3467 1848]
#                      [2178 3126]]


# Since the precision and recall is pretty low, I wil try to improve the
model by creating a balanced sample and feature scaling

#- Improving Model -#

# Feature Scaling
from sklearn.preprocessing import RobustScaler

X_N = df4[['int_rate','installment']] # split features from loan class
Y_N = df4[['loan_class']]

scaler = RobustScaler() # scale features
X_N = scaler.fit_transform(X_N)

X_N

num_neg = (Y_N == 0).sum()
num_pos = (Y_N == 1).sum()

print(num_neg)
print(num_pos)
print(round(num_pos/num_neg,2))

# Split into test and train

X_N_train, X_N_test, Y_N_train, Y_N_test = train_test_split(X_N, Y_N,
test_size=0.30, shuffle = True)

#-- verifying the training and test data --#
print(X_N_train.shape)
print(X_N_test.shape)
print(Y_N_train.shape)
print(Y_N_test.shape)
print("    ")
```

```
#-- verifying the train and test data has has 70:30 split --#
num_neg_train = (Y_N_train ==0).sum()
num_pos_train = (Y_N_train==1).sum()

num_neg_test = (Y_N_test==0).sum()
num_pos_test = (Y_N_test==1).sum()

print(num_neg_train)
print(num_pos_train)
print(round(num_neg_train/num_pos_train,2))
print("    ")
print(num_neg_test)
print(num_pos_test)
print(round(num_neg_test/num_pos_test,2))

# creating balanced sample

# import warnings

pipe = make_pipeline(SMOTE(), LogisticRegression())

#   find if a good paramter to use in logistic function

weights = np.linspace(0.005, 1.0, 100)

gsc = GridSearchCV(estimator=pipe,
                   param_grid={
                       #'smote__sampling_strategy' : [{0: int(num_neg * w) }
for w in weights]
                       #'smote__sampling_strategy':  [{0:int(num_neg)}]
                       'smote__sampling_strategy': weights
                   },
                   scoring='f1',
                   cv=5
                  )


grid_result = gsc.fit(X_N, Y_N)

print("Best parameters : %s" % grid_result.best_params_)

# Plot the weights vs f1 score
dataz = pd.DataFrame({ 'score': grid_result.cv_results_['mean_test_score'],
                       'weight': weights })
dataz.plot(x='weight')

# output: Best parameters : {'smote__sampling_strategy': 1.0}

# modeling

pipe = make_pipeline(
```

```
    SMOTE(sampling_strategy=1.0),
    LogisticRegression()
)

# Fit
pipe.fit(X_N_train, Y_N_train)

# Predict
Y_N_pred = pipe.predict(X_N_test)

# Evaluate the model
print(classification_report(Y_N_test, Y_N_pred))

# plot confusion matrix
cnf_matrix = metrics.confusion_matrix(Y_N_test, Y_N_pred)

print(cnf_matrix)

# create heatmap

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# output:
#       precision    recall   f1-score   support

#   0        0.61      0.66       0.64      5297
#   1        0.63      0.58       0.61      5322
#
# accuracy                           0.62     10619
# macro avg  0.62      0.62       0.62     10619$
# weighted avg 0.62      0.62      0.62     10619

# confusion matrix    [[3519 1778]
#                      [2232 3090]]
```

Confusion matrix

		Interpreting Confusion Matrix		
		PREDICTED		
ACTUAL		0	1	Total
	0	True Negative	False Positive	Actual negatives
	1	False Negative	True Positive	Actual positives
	Total	Predicted Negatives	Predicted Positives	

Recall

Precision

recall	Of all the loans that actually delinquinted how many could we identify
precision	Of all the loans that we predicted would delinquient how many actually delinquinted
f1 score	The F1 score can be interpreted as a weighted average of the precision and recall values, where an F1 score reaches its best value at 1 and worst value at 0.