MEF UNIVERSITY

CUSTOMER CREDIT DELINQUENCY PREDICTION

Capstone Project

Aykut Ülgenalp

İSTANBUL, 2018

MEF UNIVERSITY

CUSTOMER CREDIT DELINQUENCY PREDICTION

Capstone Project

Aykut Ülgenalp

Advisor: Dr. Berk Orbay

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Customer Credit Delinquency Prediction
Name/Last Name of the Student: Aykut Ülgenalp
Date of Thesis Defense: __/__/____

I hereby state that the graduation project prepared by Aykut Ülgenalp has been completed under my supervision. I accept this work as a "Graduation Project".

__/__/____

Dr. Berk Orbay

I hereby state that I have examined this graduation project by Aykut Ülgenalp which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

__/__/____

Prof. Dr. Özgür Özlük

Director
of
Big Data Analytics Program

We hereby state that we have held the graduation examination of _____ and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Dr. Berk Orbay	……..
2. Prof. Dr. Özgür Özlük	……..

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Aykut Ülgenalp

_____

Name Date Signature

EXECUTIVE SUMMARY

CUSTOMER CREDIT DELINQUENCY PREDICTION

Aykut Ülgenalp

Advisor: Dr. Berk Orbay

SEPTEMBER, 2018, 16 Pages

This study presents a supervised machine learning algorithm to detect delinquency probability of the customers in next 2 years with a dataset that is extracted from a leading online data source website. The aim is to predict customer delinquency before the credit is being delinquent and thus, avoid the bank from unexpected loss and also detecting to customers' possible payment difficulty and support them to pay their debts regularly. The supervised machine learning algorithms provided the opportunity to detect similarities between customer behaviours and differ them into groups for taking early actions.

Key Words: Delinquency, Credit Delinquency, Customer Delinquency, Delinquency Prediction

ÖZET

MÜŞTERİ KREDİ GECİKME TAHMİNİ

Aykut Ülgenalp

Tez Danışmanı: Dr. Berk Orbay

EYLÜL, 2018, 16 Sayfa

Bu çalışmada internette önde gelen veri kaynağı sitesinden elde edilen bir veri üzerinde, gözetimli makine öğrenmesi algoritmaları kullanılarak müşteri kredilerinde 2 yıl içerisinde yaşanması muhtemel gecikmeyi tahmin eden bir model sunulmuştur. Amaç, müşterilerin gecikmelerinden önce tahmin edebilmek ve böylece, bankayı beklenmeyen kayıptan kurtarabilmek ve ayrıca müşterilerin olası ödeme güçlüklerini tespit edip onlara ödemelerini düzgün yapabilmeleri için destek olabilmektir. Gözetimli makine öğrenmesi algoritmaları, müşteri davranışlarındaki benzerlikleri tespit etmeye ve onları farklı gruplara ayırarak erken aksiyon alabilmeye olanak sağlamaktadır.

Anahtar Kelimeler: Gecikme, Kredi Gecikmesi, Müşteri Gecikmesi, Gecikme Tahmini

TABLE OF CONTENTS

.

1. INTRODUCTION

This study examined the detection of early customer credit delinquency warnings and creating a prediction model to handle with this delinquency problem before it occurs. If the model prediction power is enough to separate bad and good customers before 2 years, it can be very helpful to avoid unexpected loss for the bank. In this chapter, the objectives and scope of the study are presented with supporting literature review.

1.1. Customer/Credit Delinquency Models: A Brief Literature Survey

In a lifetime cycle of credit, the process is briefly as follows; customer credit application, evaluation of the application, disbursement of the credit, and monitoring the behaviour of the customer before delinquency, if the delinquency situation occurs, immediately taking early collection actions, mid-collection actions, late collection actions and if the bank cannot being provide paying the dept, assume the credit non-performing and send the file legal action departments, and still cannot provide the customer pay their depts, restructure the dept or in worst scenario selling the dept to lending companies.

The final step of this cycle is the most undesired action for a bank and it is causing not only reduced profit margins but also significant sales losses for retail companies. For this reason, delinquency prediction models is one of the most important asset for a bank. This is also clearly defined from Sung Ho Ha and Ramayya Krishnan (2010) as below:

"The recent economic crisis not only reduces the profit of retailer stores but also incurs the significant losses caused by increasing the late-payment rate of credit cards. Under this pressure, the scope of credit prediction needs to be broadened to the customer management after delinquency occurs."

It is also very important for the customers, because the customers need to assist for their payment difficulties in before collection and early collection phases. If a bank has powerful manoeuvrability to take action and help the customers before the situation getting worse, most of the customers can handle their situations in a good way.

And this situation represented in a short and direct way from Fair Isaac Corp (2007) as below:

"The present invention relates generally to the optimization of strategies for collecting and recovering on delinquent debt accounts, and more particularly, to an

automated system that uses predictive modelling to optimize the use of various collection resources on a portfolio of delinquent debt accounts, including for example credit card accounts."

1.2. About The Data

In the dataset, there are 150,000 rows and 12 fields like product utilization ratios, demographic and behaviour variables which is also using to predict business model in many banks or lending companies. The inputs and their meanings are introduced as below (prepared by data owner and shared in bigml.com):

ID: ID of borrower.

SeriousDlqin2yrs: Person experienced 90 days past due delinquency or worse in 2 years (Type: Y/N). **The Target Variable**

RevolvingUtilizationOfUnsecuredLines: Total balance on credit cards and personal lines of credit except real estate and no instalment debt like car loans divided by the sum of credit limits (Type: percentage)

Age: Age of borrower in years (Type: integer)

NumberOfTime30-59DaysPastDueNotWorse: Number of times borrower has been 30-59 days past due but no worse in the last 2 years. (Type: integer).

DebtRatio: Monthly debt payments, alimony, living costs divided by monthly gross income (Type: integer)

MonthlyIncome: Monthly income (Type: real)

NumberOfOpenCreditLinesAndLoans: Number of Open loans (instalment like car loan or mortgage) and Lines of credit (e.g. credit cards) (Type: integer)

NumberOfTimes90DaysLate: Number of times borrower has been 90 days or more past due. (Type: integer)

NumberRealEstateLoansOrLines: Number of mortgage and real estate loans including home equity lines of credit (Type: integer)

NumberOfTime60-89DaysPastDueNotWorse: Number of times borrower has been 60-89 days past due but no worse in the last 2 years. (Type: integer)

NumberOfDependents: Number of dependents in family excluding themselves (spouse, children etc.). (Type: integer)

2. PROJECT STATEMENT AND METHODOLOGY

In this section, the objective and scope of the project are discussed by highlighting the business priorities. Following that, the methodology is presented covering steps such as exploratory data analysis and model deployment.

2.1. Problem Statement

As mentioned in the introduction, customer credit delinquency harms the bank in many ways, so in this project the problem is predict whether a person experience 90 day past due delinquency or worse in the next 2 years or not before they get delinquent.

The model prediction power is quite strong, so it means that the model found the similarities between bad and good customers within it, and separate and sort them in both statistical and business wise way.

Before the model works in a bank, credit monitoring system is administrated via basic reports conclusions or experienced base decisions. So, it can lead the bank to irrecoverable situations.

2.1.1 Problem Objectives

The main object is to build model that borrowers can use to help make the best financial decisions and using these models into decision making, monitoring or collection systems of the bank.

If it can be predicted before the customer is delinquent, then the bank can minimize the probability of default of the customers, number of lost customer, money and credibility.

2.1.2 Project Scope

In this project, readers can easily realise that the final model inputs are enough to predict delinquency in 2 years and how accurately they are. So, they can use same inputs in their workstream thus they do not need to try to create new complex inputs and do not waste

their times with doing complex analysis or giving clue them to creating similar models to improve their credit life cycle.

In the scope of the project, giving importance order of the inputs or the way to improve input content or telling direct way to use these model output in the reader workstream are not included. However, they can inspire or think over from the results, methodology and input contents, so they can improve this model up to higher level.

2.2. Methodology

In this project, machine learning algorithms are used which are decision tree, logistic regression, gradient boosting and random forest. Supervised machine learning algorithms is selected because there is label to represent the population are already delinquent or not. Also, there is no segmentation problem in this project, there is binary classification problem in here.

Four different supervised machine learning algorithms are created to compare those results and see the difference between them and pros and cons of these modelling techniques.

The comparing the model performances is done via statistical indicators also making the model implementation more practical and easily repeatable and explicable to business unit is also seriously considering. Because if the business unit cannot understand the way of using inputs, then implementing the model can be very hard in a bank.

For doing analysis and creating models, python language and Microsoft Azure ML Studio Tool is used, and all codes and Diagram are shown in appendix.

The data is taken from kind of a dataset source called “bigml.com”. This dataset is selected because the inputs are very similar which is using to solve same problems in many banks or lending companies.

3. DATA PREPARATION AND FEATURE SELECTION

3.1. Exploratory Data Analysis (EDA)

In this part, deep analysis of the data and variables is done. Target ratio, variables values distributions are also be investigated. Null value analysis is done and null values in variables, filled with statistical approach to prevent them to influence modelling in a bad way.

To insert the data in, “pandas” library is used, and target ratio is investigated. It is observed that the target ratio is 6.7% which is enough to creating a good predictive model. After analysed the target variable/label, all the other inputs/features are also investigated and tried to observe if there any anomalies or outliers in the input value distributions. For instance, in 'DebtRatio' input, there are some records which are greater than 1, so it is considered while creating the models.

After the describing inputs, missing values are also investigated, and for 2 inputs (‘MonthlyIncome’ , ‘NumberOfDependents’) some missing values are observed. Handling with missing values is very important to create accurate models, beyond these some modelling algorithms cannot work with missing values, so they should be managed. In this project, for deciding how they should be managed, missing values’ target ratios are analysed and tried to understand they behave different or similar to non-missing values of the input. Following, especially ‘MonthlyIncome’ input there are too much missing values and observed that missing values target ratio nearly equal the overall target ratio and rest of the data. And the other input has very few missing data, so for both inputs, missing values replaced by their average values.

3.2. Feature Selection

In this part, to create successful model, best features is selected based on correlation and feature selection algorithms.

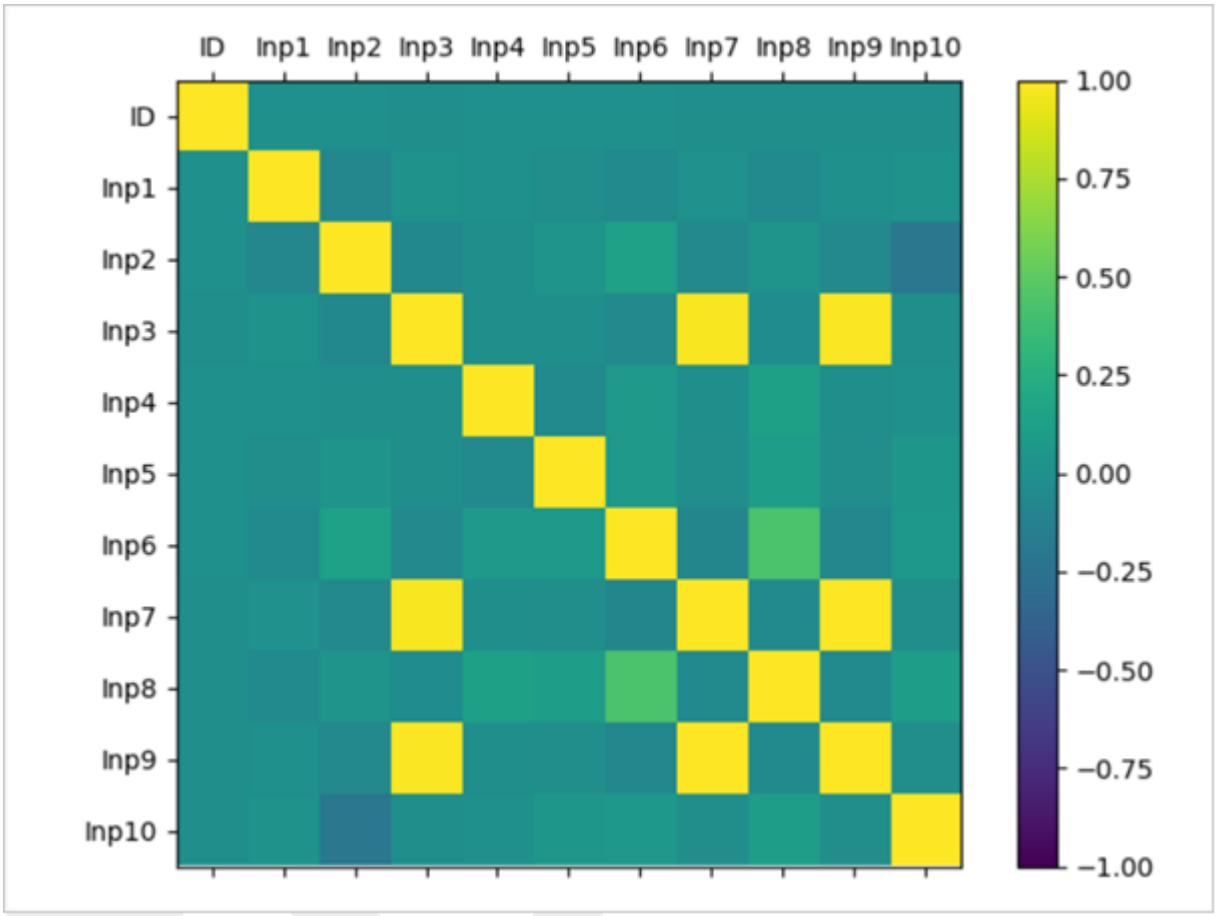
3.2.1. Correlation Matrix

In this part, correlation matrix is done and observed any correlation between inputs. And handled with correlated variables in a rational and statistical way.

Firstly, variable names are shortened to understand and read the correlation and model results easier. Variables names are shortened like below;

Long Name	Short Name
RevolvingUtilizationOfUnsecuredLines	Inp1
Age	Inp2
NumberOfTime30-59DaysPastDueNotWorse	Inp3
DebtRatio	Inp4
MonthlyIncome	Inp5
NumberOfOpenCreditLinesAndLoans	Inp6
NumberOfTimes90DaysLate	Inp7

NumberRealEstateLoansOrLines	Inp8
NumberOfTime60-89DaysPastDueNotWorse	Inp9
NumberOfDependents	Inp10
SeriousDlqin2yrs	Target_Inp



As a result of correlation analysis, some correlation is observed between features.

There are high correlation between these pair of inputs;

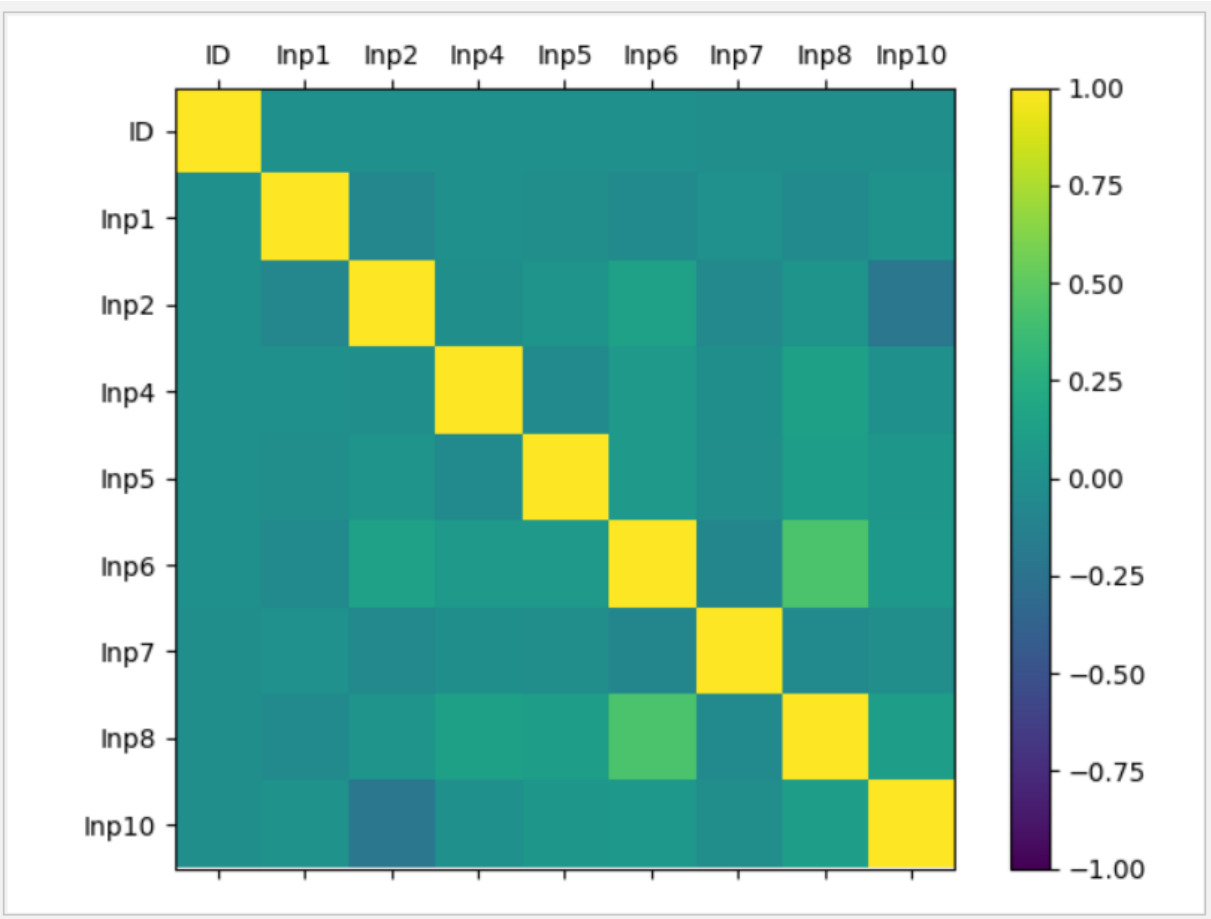
1-) inp3 and inp7

2-) inp3 and inp9

3-) inp7 and inp9

So, if inp3 (NumberOfTime30-59DaysPastDueNotWorse) and inp9 (NumberOfTime60-89DaysPastDueNotWorse), then we have none-correlation input list. (The threshold is identified as 0.70)

In the below chart, it can be seen that there is no high correlated inputs left in the input list.



3.2.2. Feature Selection

In this part, 2 different feature selection techniques are applied which are percentile selection and model-based feature selection.

Both techniques are indicated same results which is no need to eliminate inputs more after correlation elimination. But, seeing results of these techniques, made it easier to go through the modelling phase.

4. MODEL CREATING

This part will be the most complex and comprehensive part of the project. Many modelling algorithms and different version of them are performed to predict target. There are 4 different algorithms are examined;

1. Logistic Regression
2. Decision Tree
3. Gradient Boosting
4. Random Forest

Before creating models, data is splitted into 2 parts which are train and test data with a weight of 70% and 30% respectively via using from "sklearn.model_selection" library, train_test_split function. Stratify splitting method is used because it makes the 2 parts of the data equal with respect to target ratio.

4.1. Logistic Regression

Logistic Regression is old but very strong machine learning algorithm to solve binary classification problems. Because of the selected model is logistic regression model, the logic behind this algorithm is explained as follows.

It is using logistic function also called sigmoid function to create an equation between features and labels with giving features weights (mathematically coefficients). The formula can be represent like below (Brownlee J., 2016) ;

$$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \cdots + b_kX_k$$

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}$$

and

$$logit(p) = \ln(\frac{p}{1-p})$$

As a result of logistic regression model, model score in train and test data is very good.

4.2. Decision Tree

Decision tree algorithm is a very common and one of the most understandable algorithms because of its ability to visualize the results in a straight way.

After 3 different examinations, the optimal result is found.

4.3. Gradient Boosting

Gradient Boosting is one of the most powerful technique for building predictive model. Its prediction power comes from the technique that combining different predictions and minimizing the loss function.

After 4 different examinations, the optimal result is found.

4.4. Random Forest

Random Forest is a flexible, easy to use and simple machine learning algorithm that can be used for both classification and regression tasks. It builds multiple decision trees and merges them together to get a more accurate and stable prediction.[3] (towardsdatascience- on the random forest algorithm)

After 3 different examinations, the optimal result is found.

5. MODEL PERFORMANCE ANALYSIS AND MODEL SELECTION

In this part, all the model results are evaluated and tried to select optimal one by comparing with many statistical indicators.

Firstly, for every model type, one version of them are selected and evaluated by confusion matrix, precision, recall and f1-score. These statistical indicators have meanings as below respectively;

Confusion matrix represents that the relation between actual and prediction values.

There are four outcomes of binary classification (Koehrsen, W. ,2018) ;

True positives: data points labelled as positive that are actually positive

False positives: data points labelled as positive that are actually negative

True negatives: data points labelled as negative that are actually negative

False negatives: data points labelled as negative that are actually positive

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Precision is ability of a classification model to return only relevant instances. And it can be formulable as below (Koehrsen, W. ,2018);

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

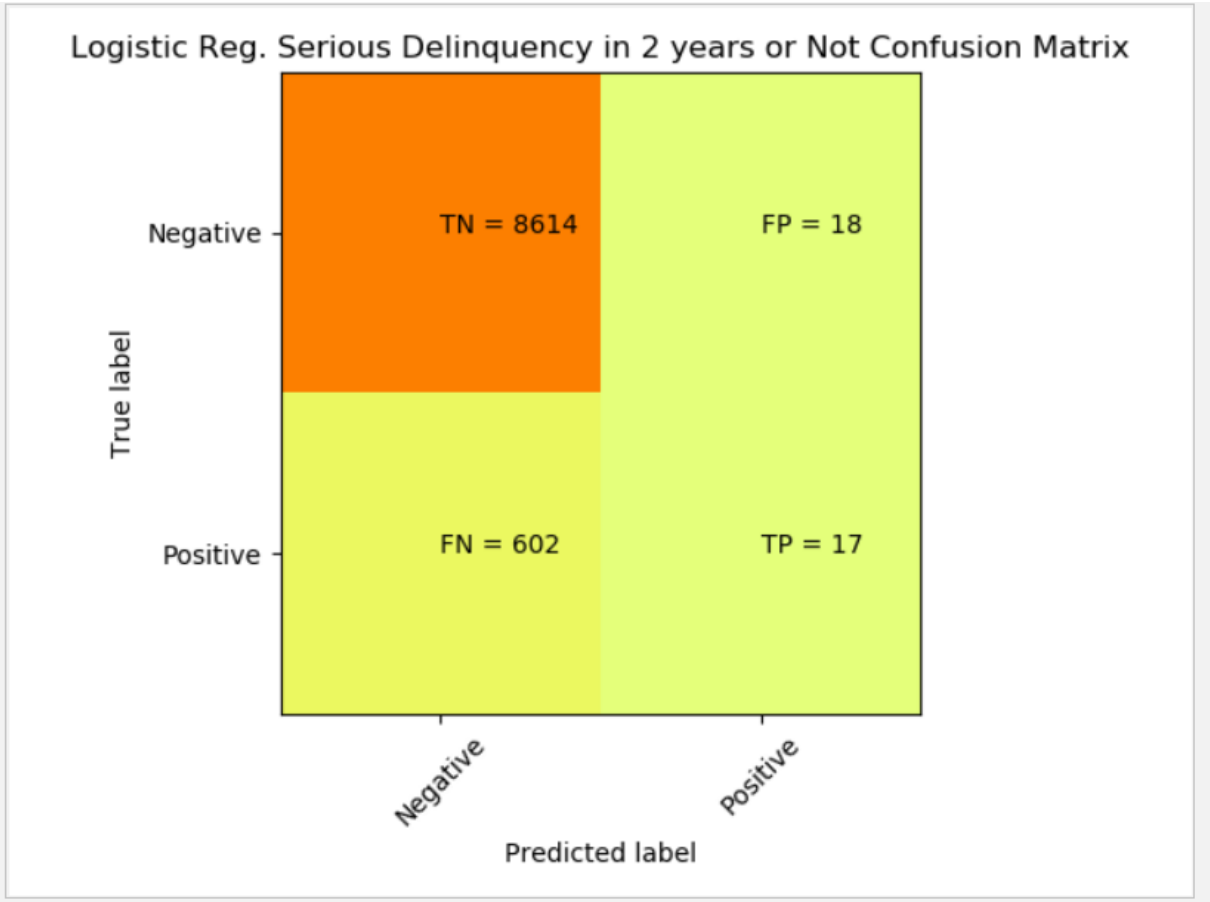
Recall is ability of a classification model to identify all relevant instances. And it can be formulable as below (Koehrsen, W. ,2018);

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

F1 score is a single metric that combines recall and precision using the harmonic mean. And it can be formulized as below (Koehrsen, W. ,2018);

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

In this project, logistic regression model is selected when these indicators are considered, and confusion matrix results in test data is as below;



And as follows, all model performance results can be found;

Model Type	Training set score	Test set score	True Positive	False Negative	False Positive	True Negative	Recall	Precision	F1 Score
Logistic Regression	0.945	0.933	15	791	13	11238	0.019	0.536	0.036
Decision Tree	0.947	0.935	128	678	107	11144	0.159	0.545	0.246
Gradient Boosting	0.949	0.935	152	654	136	11115	0.189	0.528	0.278
Random Forest	0.99	0.93	125	681	110	11141	0.155	0.532	0.24

These results are shown that, Logistic Regression is a very powerful model as well as other models, but it is more consistent in both test and train data and it predict the customer credit delinquency in 2 years very accurate way.

There are 8 features are used in the Logistic Regression model which are as below;

Long Name	Short Name	Description	Weight in Selected Model
RevolvingUtilizationOfUnsecuredLines	Inp1	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits (Type: percentage)	12,0149
Age	Inp2	Age of borrower in years (Type: integer)	-3,11605
DebtRatio	Inp4	Monthly debt payments, alimony, living costs divided by monthly gross income (Type: integer)	2,20085
MonthlyIncome	Inp5	Monthly income (Type: real)	-0,35006
NumberOfOpenCreditLinesAndLoans	Inp6	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards) (Type: integer)	-0,0739745
NumberOfTimes90DaysLate	Inp7	Number of times borrower has been 90 days or more past due. (Type: integer)	3,27848
NumberRealEstateLoansOrLines	Inp8	Number of mortgage and real estate loans including home equity lines of credit (Type: integer)	0,779585
NumberOfDependents	Inp10	Number of dependents in family excluding themselves (spouse, children etc.). (Type: integer)	1,72528
Bias	Bias	Bias	-1,39704

6. DELIVERED VALUE AND FURTHER STEPS

6.1. Project's Delivered Value

In this study, the customer credit delinquency prediction problem is solved through a machine learning algorithm to achieve utmost precision and simplicity. Following the exploratory data analysis, small number of features are eliminated which is correlated other ones, and with these 8 features a logistic regression model is developed. In the meantime, the analysis and measurement of existing algorithm's performance is presented. Taking existing process of delinquency monitoring system inadequacies into consideration, this study positively improves the performance of the prediction of customer credit delinquency in 2 years. The outcome of this project is reliable, accurate and most importantly based on statistics unlike the monitoring weekly and monthly performance reports or expert-based approach. The used algorithm is easy to understand and implement, for this reason it is very useful for bank environments. The model is open for update any time based on business side preferences.

6.2. Social and Ethical Aspects

In environments like banks or other lending companies, the model inputs must be classified, and they should not have shared with the business units. Because when a business unit know the input list entirely, they can give a clue to branch or external customers to overtake the system with intentionally or not. It can ruin the model performance and it may drag the bank to wrong path in an irreversible way. Model performance should be monitored periodically, and the performance of the model won't change in a mid-term, then model performance should be re-evaluated. Because this model also aims to change customer behaviour and teach them how should they behave when they got a credit from a bank.

6.3. Further Steps

As mentioned in the previous sections of problem statement and project scope, this study has the potential of further improvement through machine learning algorithms. The model performance is quite good and according to simplicity of the inputs, it can be very easy to understand by related managers and rest of the senior management. In further step, the number of data might be increased both in volume and variety to create more accurate prediction model. In addition to that, after increasing the data and changing the customer behaviour after 6-12 months from implementation, the model performance should be worsened naturally. In this situation, there are 2 ways to do, one of them creating similar model technique to boosting implementation process or trying other machine learning algorithms to predict customer credit delinquency in 2 years better.

REFERENCES

- Fair Isaac Corporation, Minneapolis, MN (US) (2007). Enhancing delinquent debt collection using statistical models of debt historical information and account events. Retrieved from **https://patents.google.com/patent/US7536348B2/en**

- General Electric Company, Schenectady, NY (US) (1999). Method for managing disposition of delinquent accounts. Retrieved from **https://patents.google.com/patent/US6456983B1/en**

- Ho Ha S. & Krishnan R., Pittsburg, PA (US) (2010). Predicting repayment of the credit card debt. Retrieved from **https://www.sciencedirect.com/science/article/pii/S030505481000290X?via%3Dihub#aep-keywords-id11**

- Accenture Global Services GmbH, Schauffhauen (CH) (2001). Debt collection practices. Retrieved from **https://patents.google.com/patent/US7403923B2/en**

- Koehrsen, W. (2018, March 03) Beyond Accuracy: Precision and Recall. Retrieved from **https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c**

- Donges, N. (2018, February 22) The Random Forest Algorithm. Retrieved from **https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd**

- Gorman, B. (2017, February 23) A Kaggle Master Explains Gradient Boosting. Retrieved from **http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/**

- Prashant, G. (2017, May 17) Decision Trees in Machine Learning. Retrieved from **https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052**

- Swaminathan, S. (2018, March 15) Logistic Regression - Detailed Overview. Retrieved from **https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc**

- Brownlee, J. (2016, April 1) Logistic Regression for Machine Learning. Retrieved from **https://machinelearningmastery.com/logistic-regression-for-machine-learning/**

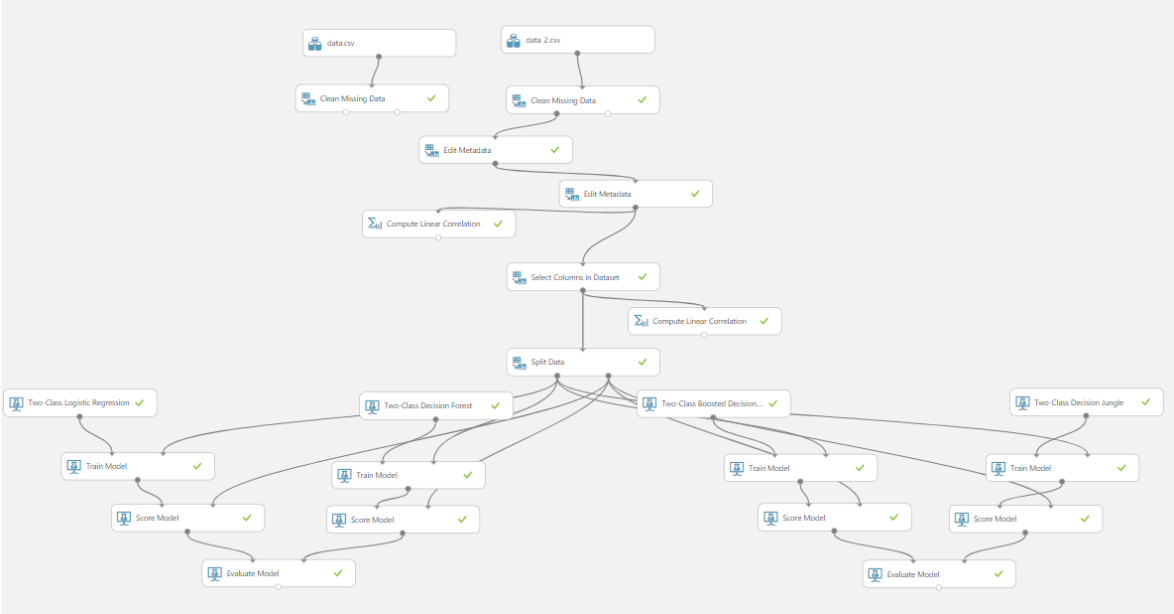
- Steele, M. (2018, January 11) Hands-On Machine Learning--Predicting Loan Delinquency. Retrieved from **https://blog.riskspan.com/hands-on-machine-learning-predicting-loan-delinquency**

- "Give Me Some Credit" dataset. Retrieved from bigML.com and kaggle.com

  https://bigml.com/user/jbosca/gallery/dataset/5a7def3d2a83476e09000456
  https://www.kaggle.com/c/GiveMeSomeCredit/download/cs-training.csv

APPENDIX

Microsoft Azure Machine Learning Studio Modelling Diagram



Codes

EDA

```
import os

os.chdir("D:\Kişisel\BIG DATA\Capstone_Project")


import pandas as pd

my_data = pd.read_csv('data.csv', sep=',')


my_data.head(15)
'''
    ID          ...           NumberOfDependents
0    1          ...                          2.0
1    2          ...                          1.0
2    3          ...                          0.0
3    4          ...                          0.0
4    5          ...                          0.0
5    6          ...                          1.0
6    7          ...                          0.0
7    8          ...                          0.0
8    9          ...                          NaN
9   10          ...                          2.0
10  11          ...                          0.0
11  12          ...                          2.0
12  13          ...                          2.0
13  14          ...                          2.0
14  15          ...                          0.0
[15 rows x 12 columns]
'''

my_data[1:1]

"""
Columns: [ID, SeriousDlqin2yrs, RevolvingUtilizationOfUnsecuredLines,
Age, NumberOfTime30-59DaysPastDueNotWorse, DebtRatio, MonthlyIncome,
NumberOfOpenCreditLinesAndLoans, NumberOfTimes90DaysLate,
NumberRealEstateLoansOrLines, NumberOfTime60-89DaysPastDueNotWorse,
NumberOfDependents]
"""

my_data.shape
##(150000, 12)



##Descriptive analysis

##Firstly let's look at our target input which is SeriousDlqin2yrs

my_data['SeriousDlqin2yrs'].describe()
```

```
"""
count    150000.000000
mean          0.066840
std           0.249746
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000

dtype: float64
"""

##As we can see in here our target rate is 6.7% which is enough to
creating a predictive model

##And let's continue to describing the other inputs

my_data['RevolvingUtilizationOfUnsecuredLines'].describe()

"""
count    150000.000000
mean          6.048438
std         249.755371
min           0.000000
25%           0.029867
50%           0.154181
75%           0.559046
max       50708.000000

dtype: float64
"""

my_data['Age'].describe()

"""
count    150000.000000
mean         52.295207
std          14.771866
min           0.000000
25%          41.000000
50%          52.000000
75%          63.000000
max         109.000000
Name: Age, dtype: float64
"""

##We will working on a population with old age, It can be a helpful
information to understand our model outputs

my_data['NumberOfTime30-59DaysPastDueNotWorse'].describe()

"""
count    150000.000000
mean          0.421033
std           4.192781
min           0.000000
25%           0.000000
50%           0.000000
```

```
75%            0.000000
max           98.000000
Name: NumberOfTime30-59DaysPastDueNotWorse, dtype: float64
"""

my_data['DebtRatio'].describe()

"""
count    150000.000000
mean        353.005076
std        2037.818523
min           0.000000
25%           0.175074
50%           0.366508
75%           0.868254
max      329664.000000
Name: DebtRatio, dtype: float64
"""

my_data[my_data['DebtRatio']>1].count()
##DebtRatio   35137

#as we can see here, there are 35137(23%) borrowers has a debtratio
greater than 1. It is a huge number of rows. So we have to taking account
this situation while creating model

my_data['MonthlyIncome'].describe()
"""
my_data['MonthlyIncome'].describe()
count    1.202690e+05
mean     6.670221e+03
std      1.438467e+04
min      0.000000e+00
25%      3.400000e+03
50%      5.400000e+03
75%      8.249000e+03
max      3.008750e+06
Name: MonthlyIncome, dtype: float64
"""

(my_data.MonthlyIncome)[:50].describe()

"""
count       45.000000
mean      7139.266667
std       9892.955353
min          0.000000
25%       2600.000000
50%       3661.000000
75%       8800.000000
max      63588.000000
Name: MonthlyIncome, dtype: float64
"""

##as we can see here when we have some NA rows, descriptive functions
doesn't work true

my_data['NumberOfOpenCreditLinesAndLoans'].describe()
```

```
"""
count    150000.000000
mean          8.452760
std           5.145951
min           0.000000
25%           5.000000
50%           8.000000
75%          11.000000
max          58.000000
Name: NumberOfOpenCreditLinesAndLoans, dtype: float64
"""

my_data['NumberOfTimes90DaysLate'].describe()

"""
count    150000.000000
mean          0.265973
std           4.169304
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          98.000000
Name: NumberOfTimes90DaysLate, dtype: float64
"""

my_data['NumberRealEstateLoansOrLines'].describe()

"""
count    150000.000000
mean          1.018240
std           1.129771
min           0.000000
25%           0.000000
50%           1.000000
75%           2.000000
max          54.000000
Name: NumberRealEstateLoansOrLines, dtype: float64
"""

my_data['NumberOfTime60-89DaysPastDueNotWorse'].describe()

"""
count    150000.000000
mean          0.240387
std           4.155179
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          98.000000
Name: NumberOfTime60-89DaysPastDueNotWorse, dtype: float64
"""

my_data['NumberOfDependents'].describe()

"""
count    146076.000000
mean          0.757222
```

```
std            1.115086
min            0.000000
25%            0.000000
50%            0.000000
75%            1.000000
max           20.000000
Name: NumberOfDependents, dtype: float64
"""

##Let's look at number of NA counts

##NULL ANALYSIS AND HANDLING WITH THEM###
#NULL row control

my_data.isnull().sum()
"""
ID                                          0
SeriousDlqin2yrs                            0
RevolvingUtilizationOfUnsecuredLines        0
Age                                         0
NumberOfTime30-59DaysPastDueNotWorse        0
DebtRatio                                   0
MonthlyIncome                           29731
NumberOfOpenCreditLinesAndLoans             0
NumberOfTimes90DaysLate                     0
NumberRealEstateLoansOrLines                0
NumberOfTime60-89DaysPastDueNotWorse        0
NumberOfDependents                       3924
"""

##as we can see in here there are some NA rows in only 2 inputs


#as we can see here, there too much NA rows in 'MonthlyIncome' input. So
we cannot delete these rows, because we can lose too much data.
#so we have 2 options, we can delete this input from our dataset or we
can replace NA's into the mean of the total 'MonthlyIncome'
#But replacing mean value is a little bit dangerous way, because we
manipulating the data and model can learn very wrong from these data.
#Let's look the mean of 'MonthlyIncome'

my_data['MonthlyIncome'].mean()
##6670.221237392844

my_data['MonthlyIncome'].max()
##3008750.0

(my_data[my_data['MonthlyIncome'].isnull()]).groupby(['SeriousDlqin2yrs']
).size()
"""
SeriousDlqin2yrs
0    28062
1     1669
dtype: int64
"""
##so we can see that the target ratio of NA rows of  is 'MonthlyIncome'
is nearly 6%. So it is very similar to our total target ratio 6.7%
##Then we can change assume that these NA rows are similar to other rows,
```

```
then we can replace them as the average.

##alternative way to find NA rows target ratio
"""
b=my_data[my_data['MonthlyIncome'].isnull()]
c=pd.merge(my_data,b,on=['ID'])
c.groupby(['SeriousDlqin2yrs_y']).size()
0    28062
1     1669
"""


##replacing NA's their's mean value

my_data.mean()

"""
ID                                     75000.500000
SeriousDlqin2yrs                           0.066840
RevolvingUtilizationOfUnsecuredLines       6.048438
Age                                       52.295207
NumberOfTime30-59DaysPastDueNotWorse       0.421033
DebtRatio                                353.005076
MonthlyIncome                           6670.221237
NumberOfOpenCreditLinesAndLoans            8.452760
NumberOfTimes90DaysLate                    0.265973
NumberRealEstateLoansOrLines               1.018240
NumberOfTime60-89DaysPastDueNotWorse       0.240387
NumberOfDependents                         0.757222
dtype: float64
"""

my_data_2=my_data.fillna(my_data.mean())


my_data_2.mean()

"""
ID                                     75000.500000
SeriousDlqin2yrs                           0.066840
RevolvingUtilizationOfUnsecuredLines       6.048438
Age                                       52.295207
NumberOfTime30-59DaysPastDueNotWorse       0.421033
DebtRatio                                353.005076
MonthlyIncome                           6670.221237
NumberOfOpenCreditLinesAndLoans            8.452760
NumberOfTimes90DaysLate                    0.265973
NumberRealEstateLoansOrLines               1.018240
NumberOfTime60-89DaysPastDueNotWorse       0.240387
NumberOfDependents                         0.757222
dtype: float64
"""

my_data_2.isnull().sum()

"""
ID                                     0
SeriousDlqin2yrs                       0
RevolvingUtilizationOfUnsecuredLines   0
```

```
Age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
MonthlyIncome                           0
NumberOfOpenCreditLinesAndLoans         0
NumberOfTimes90DaysLate                 0
NumberRealEstateLoansOrLines            0
NumberOfTime60-89DaysPastDueNotWorse    0
NumberOfDependents                      0
dtype: int64
"""
##So as we can see here, the mean of the inputs did not change and now
there are no NA rows in our data


##Let's look at the correlation between inputs

my_data_model = pd.read_csv('data_2.csv', sep=',')

my_data_model_2=my_data_model.fillna(my_data_model.mean())

target=my_data_model_2.iloc[:,-1:]
labels=my_data_model_2.iloc[:,:-1]
```

#CORRELATION

```
import matplotlib.pyplot as plt
import numpy
correlations = labels.corr()
# plotting correlation matrix
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,11,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(labels.head(0))
ax.set_yticklabels(labels.head(0))
plt.show()



#as we can see in the matrix there are high correlation between

### 1-) inp3 and inp7
### 2-) inp3 and inp9
### 3-) inp7 and inp9

# so if we eliminate inp3(NumberOfTime30-59DaysPastDueNotWorse) and
inp9(NumberOfTime60-89DaysPastDueNotWorse) then we have none-correlation
input list

labels_new=labels.drop(columns=['Inp3', 'Inp9'])
labels_new.head(0)

"""
```

```
Columns: [ID, Inp1, Inp2, Inp4, Inp5, Inp6, Inp7, Inp8, Inp10]
"""

##let's look at the correlation again and making sure there is no
correlations between inputs


correlations2 = labels_new.corr()
# plotting correlation matrix
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations2, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(labels_new.head(0))
ax.set_yticklabels(labels_new.head(0))
plt.show()



#and we can see that there is no high correlations now

##Let's creating some models and try to understand prediction power with
these inputs
```

DATA ANALYSIS

```
####DATASPLITTING####

from sklearn.model_selection import train_test_split

y=target.Target_Inp
X=labels_new

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state=10)

####DATA SCALING####

from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)

#########LOGISTIC REGRESSION#############
```

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression().fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))

"""
Training set score: 0.945
Test set score: 0.933
"""

#As we can see in here, the performance of the model is quite good with
Logistic regression algorithm


logreg2 = LogisticRegression(C=0.0001).fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg2.score(X_train,
y_train)))
print("Test set score: {:.3f}".format(logreg2.score(X_test, y_test)))

"""
Training set score: 0.945
Test set score: 0.933
"""

#as we can see in here it makes no difference to change C parameters.


##########DECISION TREE##########

from sklearn.tree import DecisionTreeClassifier

tree1 = DecisionTreeClassifier(random_state=0)
tree1.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree1.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(tree1.score(X_test, y_test)))
```

```
"""
Accuracy on training set: 1.000
Accuracy on test set: 0.901
"""



tree2 = DecisionTreeClassifier(max_depth=4,random_state=0)
tree2.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree2.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(tree2.score(X_test, y_test)))

"""
Accuracy on training set: 0.947
Accuracy on test set: 0.935
"""

tree3 = DecisionTreeClassifier(max_depth=5,random_state=0)
tree3.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree3.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(tree3.score(X_test, y_test)))

"""
Accuracy on training set: 0.947
Accuracy on test set: 0.933
"""


######RANDOM FOREST######

from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(forest.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

```
"""
Accuracy on training set: 1.000
Accuracy on test set: 0.935
"""

forest2 = RandomForestClassifier(n_estimators=5, random_state=0)
forest2.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(forest2.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(forest2.score(X_test,
y_test)))

"""
Accuracy on training set: 0.990
Accuracy on test set: 0.930
"""


forest3 = RandomForestClassifier(n_estimators=4, random_state=0)
forest3.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(forest3.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(forest3.score(X_test,
y_test)))

"""
Accuracy on training set: 0.981
Accuracy on test set: 0.932
"""


###########GRADIENT BOOSTING#######

from sklearn.ensemble import GradientBoostingClassifier

gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train,
```

```
y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))

"""
Accuracy           on           training           set:           0.948
Accuracy            on            test            set:           0.936
"""


gbrt2    =    GradientBoostingClassifier(random_state=0,    max_depth=4)
gbrt2.fit(X_train,                                                y_train)
print("Accuracy  on  training  set:  {:.3f}".format(gbrt2.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt2.score(X_test, y_test)))

"""
Accuracy           on           training           set:           0.949
Accuracy            on            test            set:           0.935
"""


gbrt3    =    GradientBoostingClassifier(random_state=0,    max_depth=5)
gbrt3.fit(X_train,                                                y_train)
print("Accuracy  on  training  set:  {:.3f}".format(gbrt3.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt3.score(X_test, y_test)))

"""
Accuracy           on           training           set:           0.951
Accuracy            on            test            set:           0.934
"""


gbrt4  =  GradientBoostingClassifier(random_state=0,  learning_rate=0.01)
gbrt4.fit(X_train,                                                y_train)
print("Accuracy  on  training  set:  {:.3f}".format(gbrt4.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt4.score(X_test, y_test)))
```

```
"""
Accuracy on training set: 0.945
Accuracy on test set: 0.933
"""

#######FEATURE SELECTION#######

#We will try 2 different feature selection. Let's start with percentile selection

#1-PERCENTILE SELECTION#

from sklearn.feature_selection import SelectPercentile

select_perc = SelectPercentile(percentile=50)
select_perc.fit(X_train, y_train)
# transform training set
X_train_selected_perc = select_perc.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected_perc.shape:
{}".format(X_train_selected_perc.shape))

"""
X_train.shape: (137943, 9)
X_train_selected_perc.shape: (137943, 4)

"""



logreg.fit(X_train, y_train)
print("Score with all features: {:.3f}".format(logreg.score(X_test,
y_test)))
##Score with all features: 0.933
X_test_selected_perc = select_perc.transform(X_test)
logreg.fit(X_train_selected_perc, y_train)
print("Score with only selected features:
{:.3f}".format(logreg.score(X_test_selected_perc, y_test)))
```

```
##Score with only selected features: 0.933

##In logistic regression, there is no difference. Let's try for decision tree

print("Score with all features: {:.3f}".format(tree1.score(X_test, y_test)))
##Score with all features: 0.901
tree1.fit(X_train_selected_perc, y_train)
print("Score with only selected features: {:.3f}".format(tree1.score(X_test_selected_perc, y_test)))
##Score with only selected features: 0.933

#As you can see in here, when eliminate some inputs, decision tree's performance is increased.

#So let's continue with the other models

print("Score with all features: {:.3f}".format(tree2.score(X_test, y_test)))
##Score with all features: 0.935
tree2.fit(X_train_selected_perc, y_train)
print("Score with only selected features: {:.3f}".format(tree2.score(X_test_selected_perc, y_test)))
##Score with only selected features: 0.934
##Increased again


print("Score with all features: {:.3f}".format(tree3.score(X_test, y_test)))
##Score with all features: 0.933
tree3.fit(X_train_selected_perc, y_train)
print("Score with only selected features: {:.3f}".format(tree3.score(X_test_selected_perc, y_test)))
##Score with only selected features: 0.935
##Increased again


print("Score with all features: {:.3f}".format(forest.score(X_test,
```

```
y_test)))
##Score           with           all           features:           0.933
forest.fit(X_train_selected_perc,                                   y_train)
print("Score         with         only         selected         features:
{:.3f}".format(forest.score(X_test_selected_perc,                 y_test)))
##Score       with       only       selected       features:       0.935
##Increased                                                        again



print("Score  with  all  features:  {:.3f}".format(forest2.score(X_test,
y_test)))
##Score           with           all           features:           0.930
forest2.fit(X_train_selected_perc,                                  y_train)
print("Score         with         only         selected         features:
{:.3f}".format(forest2.score(X_test_selected_perc,                y_test)))
##Score       with       only       selected       features:       0.934
##Increased                                                        again


print("Score  with  all  features:  {:.3f}".format(forest3.score(X_test,
y_test)))
##Score           with           all           features:           0.931
forest3.fit(X_train_selected_perc,                                  y_train)
print("Score         with         only         selected         features:
{:.3f}".format(forest3.score(X_test_selected_perc,                y_test)))
##Score       with       only       selected       features:       0.933
##Increased                                                        again


print("Score   with   all   features:   {:.3f}".format(gbrt.score(X_test,
y_test)))
##Score           with           all           features:           0.932
gbrt.fit(X_train_selected_perc,                                     y_train)
print("Score         with         only         selected         features:
{:.3f}".format(gbrt.score(X_test_selected_perc,                   y_test)))
##Score       with       only       selected       features:       0.935
##Increased                                                        again
```

```
print("Score with all features: {:.3f}".format(gbrt2.score(X_test,
y_test)))
##Score with all features: 0.935
gbrt2.fit(X_train_selected_perc, y_train)
print("Score with only selected features:
{:.3f}".format(gbrt2.score(X_test_selected_perc, y_test)))
##Score with only selected features: 0.935
##stable

gbrt3.fit(X_train,y_train)
print("Score with all features: {:.3f}".format(gbrt3.score(X_test,
y_test)))
##Score with all features: 0.934
gbrt3.fit(X_train_selected_perc, y_train)
print("Score with only selected features:
{:.3f}".format(gbrt3.score(X_test_selected_perc, y_test)))
##Score with only selected features: 0.935
##Increased again


print("Score with all features: {:.3f}".format(gbrt4.score(X_test,
y_test)))
##Score with all features: 0.933
gbrt4.fit(X_train_selected_perc, y_train)
print("Score with only selected features:
{:.3f}".format(gbrt4.score(X_test_selected_perc, y_test)))
##Score with only selected features: 0.933
##Stable


#2-Model-based Feature Selection#

from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select_model = SelectFromModel(
    RandomForestClassifier(n_estimators=100, random_state=42),
    threshold="median")
```

```
select_model.fit(X_train, y_train)
X_train_selected_model = select_model.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected_model.shape:
{}".format(X_train_selected_model.shape))

"""
X_train.shape: (137943, 9)
X_train_selected_model.shape: (137943, 5)
"""

logreg.fit(X_train, y_train)
X_test_selected_model = select_model.transform(X_test)
print("Score with all features: {:.3f}".format(logreg.score(X_test,
y_test)))
##Score with all features: 0.933

logreg.fit(X_train_selected_model, y_train)
print("Score with only selected features:
{:.3f}".format(logreg.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.933

##In logistic regression, there is no difference. Let's try for decision
tree

print("Score with all features: {:.3f}".format(tree1.score(X_test,
y_test)))
##Score with all features: 0.901
tree1.fit(X_train_selected_model, y_train)
print("Score with only selected features:
{:.3f}".format(tree1.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.893

#As you can see in here, when eliminate some inputs, decision tree's
performance is decreased. So with selected features
# based on randomforest model is not performed as well as whole feature
set.
```

```
#So let's continue with the other models

print("Score with all features: {:.3f}".format(tree2.score(X_test,
y_test)))
##Score with all features: 0.935
tree2.fit(X_train_selected_model, y_train)
print("Score with only selected features:
{:.3f}".format(tree2.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.933
##Decreased again

print("Score with all features: {:.3f}".format(tree3.score(X_test,
y_test)))
##Score with all features: 0.933
tree3.fit(X_train_selected_model, y_train)
print("Score with only selected features:
{:.3f}".format(tree3.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.934
##Increased this time

print("Score with all features: {:.3f}".format(forest.score(X_test,
y_test)))
##Score with all features: 0.933
forest.fit(X_train_selected_model, y_train)
print("Score with only selected features:
{:.3f}".format(forest.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.933
##Stable

print("Score with all features: {:.3f}".format(forest2.score(X_test,
y_test)))
##Score with all features: 0.930
forest2.fit(X_train_selected_model, y_train)
print("Score with only selected features:
```

```
{:.3f}".format(forest2.score(X_test_selected_model,             y_test)))
##Score       with       only       selected       features:        0.928
##Decreased


print("Score  with  all  features:  {:.3f}".format(forest3.score(X_test,
y_test)))
##Score           with           all           features:           0.931
forest3.fit(X_train_selected_model,                                y_train)
print("Score         with         only        selected         features:
{:.3f}".format(forest3.score(X_test_selected_model,             y_test)))
##Score       with       only       selected       features:        0.930
##Decreased                                                        again


print("Score   with   all   features:   {:.3f}".format(gbrt.score(X_test,
y_test)))
##Score           with           all           features:           0.932
gbrt.fit(X_train_selected_model,                                   y_train)
print("Score         with         only        selected         features:
{:.3f}".format(gbrt.score(X_test_selected_model,                y_test)))
##Score       with       only       selected       features:        0.934
##Increased

gbrt2.fit(X_train,y_train)
print("Score  with  all  features:  {:.3f}".format(gbrt2.score(X_test,
y_test)))
##Score           with           all           features:           0.935
gbrt2.fit(X_train_selected_model,                                  y_train)
print("Score         with         only         selected         features:
{:.3f}".format(gbrt2.score(X_test_selected_model,               y_test)))
##Score       with       only       selected       features:        0.933
##Decreased


print("Score  with  all  features:  {:.3f}".format(gbrt3.score(X_test,
y_test)))
##Score           with           all           features:           0.934
gbrt3.fit(X_train_selected_model,                                  y_train)
```

```
print("Score with only selected features: {:.3f}".format(gbrt3.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.934
##Stable


print("Score with all features: {:.3f}".format(gbrt4.score(X_test, y_test)))
##Score with all features: 0.933
gbrt4.fit(X_train_selected_model, y_train)
print("Score with only selected features: {:.3f}".format(gbrt4.score(X_test_selected_model, y_test)))
##Score with only selected features: 0.933
##Stable




####MODEL SELECTION#####

##Based on the accuracy score, I selected one version from every model types

# Confusion matrices

logreg.fit(X_train,y_train)
y_predicted_logreg = logreg.predict(X_test)

tree2.fit(X_train,y_train)
y_predicted_tree2 = tree2.predict(X_test)

forest2.fit(X_train_selected_perc,y_train)
y_predicted_forest2 = forest2.predict(X_test_selected_perc)

gbrt2.fit(X_train,y_train)
y_predicted_gbrt2 = gbrt2.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
print("Logistic Regression:")
print(confusion_matrix(y_test, y_predicted_logreg))
print("\nDecision Tree:")
print(confusion_matrix(y_test, y_predicted_tree2))
print("\nRandom Fores:")
print(confusion_matrix(y_test, y_predicted_forest2))
print("\nGradient Boosting")
print(confusion_matrix(y_test, y_predicted_gbrt2))

"""
Logistic Regression:
[[8593 39]
 [ 562 57]]
Decision Tree:
[[8583 49]
 [ 554 65]]
Random Fores:
[[8567 65]
 [ 544 75]]
Gradient Boosting
[[8593 39]
 [ 562 57]]
"""

# Precision, recall and f-score
from sklearn.metrics import f1_score
print("f1 score Logistic Regression: {:.2f}".format(f1_score(y_test,
y_predicted_logreg)))
print("f1 score Decision Tree: {:.2f}".format(f1_score(y_test,
y_predicted_tree2)))
print("f1 score Random Forest: {:.2f}".format(f1_score(y_test,
y_predicted_forest2)))
print("f1 score Gradient Boosting: {:.2f}".format(f1_score(y_test,
y_predicted_gbrt2)))

"""
```

```
f1           score           Logistic           Regression:           0.16
f1            score            Decision            Tree:            0.18
f1            score            Random            Forest:            0.20
f1           score           Gradient           Boosting:           0.16
"""


from         sklearn.metrics         import         classification_report
print(classification_report(y_test,                  y_predicted_logreg,
                            target_names=["not              delinquent",
"delinquent"]))
"""
                precision            recall      f1-score          support
not  delinquent            0.94          1.00          0.97          8632
    delinquent            0.59          0.09          0.16           619
   avg  /  total            0.92          0.94          0.91          9251
"""

print(classification_report(y_test,                  y_predicted_tree2,
                            target_names=["not              delinquent",
"delinquent"]))

"""
                precision            recall      f1-score          support
not  delinquent            0.94          0.99          0.97          8632
    delinquent            0.57          0.11          0.18           619
   avg  /  total            0.91          0.93          0.91          9251
"""

print(classification_report(y_test,                y_predicted_forest2,
                            target_names=["not              delinquent",
"delinquent"]))

"""
                precision            recall      f1-score          support
not  delinquent            0.94          0.99          0.97          8632
    delinquent            0.54          0.12          0.20           619
   avg  /  total            0.91          0.93          0.91          9251
"""
```

```
print(classification_report(y_test,                    y_predicted_gbrt2,
                            target_names=["not                delinquent",
"delinquent"]))

"""
                precision            recall      f1-score          support
not  delinquent            0.94           1.00           0.97           8632
    delinquent            0.59           0.09           0.16            619
   avg  /  total            0.92           0.94           0.91           9251
"""

##AS WE CAN SEE IN HERE LOGISTIC REGRESSION AND GRADIENT BOOSTING MODELS
ARE                                THE                                BEST
##I               SELECTED                LOGISTIC                REGRESSION

########CONFUSION                                                MATRIX########

logreg.fit(X_train,y_train)
y_predicted_logreg                =                logreg.predict(X_test)

from           sklearn.metrics           import           confusion_matrix

cm         =         confusion_matrix(y_test,         y_predicted_logreg)
print(cm)

"""
[[8614                                                                  18]
 [              602                                                     17]]
"""

plt.clf()
plt.imshow(cm,interpolation='nearest',cmap=plt.cm.Wistia)
classNames=['Negative','Positive']
plt.title('Logistic Reg. Serious Delinquency in 2 years or Not Confusion
Matrix')
plt.ylabel('True                                                        label')
plt.xlabel('Predicted                                                   label')
tick_marks                  =                  np.arange(len(classNames))
```

```
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
plt.show()
```