**Project 1 : House Loan Data**

The Core Data Set files is used as well as has been broken into Small Files for various operation.

Breakup of the files are and its details are listed below for better understanding.

* **HomeCredit\_columns\_description.csv**

1. This file contains descriptions for the columns in the various data files.

* **application\_{train|test}.csv**

1. This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
2. Static data for all applications. One row represents one loan in our data sample.

* **bureau.csv**

1. All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
2. For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

* **bureau\_balance.csv**

1. Monthly balances of previous credits in Credit Bureau.
2. This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample \* # of relative previous credits \* # of months where we have some history observable for the previous credits) rows.

* **POS\_CASH\_balance.csv**

1. Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
2. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample \* # of relative previous credits \* # of months in which we have some history observable for the previous credits) rows.

* **credit\_card\_balance.csv**

1. Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
2. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample \* # of relative previous credit cards \* # of months where we have some history observable for the previous credit card) rows.

* **previous\_application.csv**

1. All previous applications for Home Credit loans of clients who have loans in our sample.
2. There is one row for each previous application related to loans in our data sample.

* **installments\_payments.csv**
* Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
* There is a) one row for every payment that was made plus b) one row each for missed payment.
* One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

A close up of a map

Description automatically generated

**Real World / Business Objectives and Constraints:**

Before starting any problem, it is better to lay the constraints. So that we can change the modelling process based on the constraints.

**1. No strict latency constraints.**

Given the loan application data, we don’t have to predict whether the applicant is going to repay loan in milli seconds or seconds. We can take couple of minutes to predict. Even 1 hour should be fine. Since we don’t have strict latency constraint, we can use Ensemble models like Random forest, Xgboost etc.

**2. Predict the probability of capability of each applicant of repaying a loan.**

Suppose there are two applicants A, B whose probability values of repaying loan is 0.6 and 0.9, since the general threshold each machine learning model considers is 0.5. Both the applicants are labeled as ‘Repaying the loan’. But applicant B is more likely to repay the loan when compared to applicant A. If we predict the probabilities we can set the threshold like 0.8, 0.9 based on the business requirements.

**3. The cost of a mis-classification is very high.**

Let’s say, for an applicant A our model labelled as ‘Repaying the loan’. So the organization sanctioned the loan for that applicant. But the applicant for some reason is not able to pay the loan. This type of scenarios is loss to the organization. Hence we should come up with a model which can reduce the mis-classifications as much as possible.

**4. Interpretability is partially important.**

As long as our model predicts well on test data, we don’t need to care much about interpretability. Since our problem is not related to medical domain(Interpretability is important- cancer detection), it is fine if we can give some form of interpretability like feature importances.

**Performance Metric:**

In this problem, the data is imbalanced. So we can’t use accuracy as a error metric. When data is imbalanced we can use Log loss, F1-score and AUC. Here we are sticking to AUC which can handle imbalanced datasets.

**Area Under Curve (AUC)**

An ROC curve is the most commonly used way to visualize the performance of a binary classifier, and AUC is (arguably) the best way to summarize its performance in a single number.

Confusion Matrix (To get an overview of complete predictions)

**Exploratory Data Analysis:**

At first, import the necessary packages.

import pandas as pd  
import sklearn  
import numpy as np  
import matplotlib.pyplot as plt  
import os  
import warnings  
import seaborn as snsfrom sklearn.preprocessing import OneHotEncoder  
from sklearn.impute import SimpleImputer  
from sklearn.pipeline import Pipeline  
from sklearn.compose import ColumnTransformer  
from sklearn.preprocessing import StandardScaler  
from sklearn.svm import LinearSVC  
from sklearn.metrics import roc\_auc\_score  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import roc\_auc\_score  
from sklearn.calibration import CalibratedClassifierCV  
from sklearn.metrics import confusion\_matrix  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
from sklearn.linear\_model import SGDClassifierimport plotly.offline as py  
import plotly.graph\_objs as gofrom plotly.offline import init\_notebook\_mode, iplot  
from sklearn.model\_selection import train\_test\_split  
init\_notebook\_mode(connected=True)import cufflinks as cf  
cf.go\_offline()import pickle  
import gc  
import lightgbm as lgbwarnings.filterwarnings('ignore')  
%matplotlib inline

Load the data from given csv file into a pandas dataframe.

print('Reading the data....', end='')  
application = pd.read\_csv('application\_train.csv')  
print('done!!!')  
print('The shape of data:',application.shape)  
print('First 5 rows of data:')  
application.head()

A screenshot of a social media post

Description automatically generated

**We are using ‘application\_train.csv’ file :**

1. This dataset consists of 307511 rows and 122 columns.

2. Each row has unique id ‘SK\_ID\_CURR’ and the output label is in the ‘TARGET’ column.

3. TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.

4. The description of each column can be found in the file ‘**HomeCredit\_columns\_description.csv’**

**Let us check for missing values in each column.**

count = application.isnull().sum().sort\_values(ascending=False)  
percentage = ((application.isnull().sum()/len(application)\*100)).sort\_values(ascending=False)missing\_application = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])  
print('Count and percentage of missing values for top 20 columns:')  
missing\_application.head(20)

A screenshot of a cell phone

Description automatically generated

Observations:

1. There are lot of missing values in each column.

2. We need to somehow handle these missing values, we will see how to handle later in the case study.

**Let’s check for duplicate data:**

columns\_without\_id = [col for col in application.columns if col!='SK\_ID\_CURR']#Checking for duplicates in the data.  
application[application.duplicated(subset = columns\_without\_id, keep=False)]

print('The no of duplicates in the data:',application[application.duplicated(subset = columns\_without\_id, keep=False)].shape[0])



**Let’s check the distribution of data points among output class.**

cf.set\_config\_file(theme='polar')

contract\_val = application['NAME\_CONTRACT\_TYPE'].value\_counts()  
contract\_df = pd.DataFrame({'labels': contract\_val.index,  
'values': contract\_val.values  
})

contract\_df.iplot(kind='pie',labels='labels',values='values', title='Types of Loan', hole = 0.6)

A close up of a logo

Description automatically generated

**Observations:**

1. The data is imbalanced (91.9%(Loan repayed-0) and 8.07%(Loan not repayed-1)) and we need to handle this problem.

**Distribution of AMT\_INCOME\_TOTAL.**

application[application['AMT\_INCOME\_TOTAL'] < 2000000]['AMT\_INCOME\_TOTAL'].iplot(kind='histogram', bins=100,  
xTitle = 'Total Income', yTitle ='Count of applicants',  
title='Distribution of AMT\_INCOME\_TOTAL')

A picture containing orange, sitting, light, table

Description automatically generated

(application[application['AMT\_INCOME\_TOTAL'] > 1000000]['TARGET'].value\_counts())/len(application[application['AMT\_INCOME\_TOTAL'] > 1000000])\*100



Observations:

1. The distribution is right skewed and there are extreme values, we can apply log distribution.

2. People with high income(>1000000) are likely to repay the loan.

**Types of loan available.**

cf.set\_config\_file(theme='polar')  
contract\_val = application['NAME\_CONTRACT\_TYPE'].value\_counts()  
contract\_df = pd.DataFrame({'labels': contract\_val.index,  
'values': contract\_val.values  
})  
contract\_df.iplot(kind='pie',labels='labels',values='values', title='Types of Loan', hole = 0.6)

**A screenshot of a cell phone

Description automatically generated**

Observations:

* Many people are willing to take cash loan than revolving loan

**Distribution of AMT\_CREDIT.**

application['AMT\_CREDIT'].iplot(kind='histogram', bins=100,  
xTitle = 'Credit Amount',yTitle ='Count of applicants',  
title='Distribution of AMT\_CREDIT')

A screenshot of a cell phone

Description automatically generated

np.log(application['AMT\_CREDIT']).iplot(kind='histogram', bins=100,  
xTitle = 'log(Credit Amount)',yTitle ='Count of applicants',  
title='Distribution of log(AMT\_CREDIT)')

A picture containing table, sitting, orange, computer

Description automatically generated

Observations:

1. People who are taking credit for large amount are very likely to repay the loan.

2. Originally the distribution is right skewed, we used log transformation to make it normal distributed.

**Distribution of Name of type of the Suite in terms of loan is repayed or not.**

cf.set\_config\_file(theme='polar')  
suite\_val = (application['NAME\_TYPE\_SUITE'].value\_counts()/len(application))\*100  
suite\_val.iplot(kind='bar', xTitle = 'Name of type of the Suite',  
yTitle='Count of applicants in %',  
title='Who accompanied client when applying for the application in % ')

A screenshot of a cell phone

Description automatically generated

suite\_val = application['NAME\_TYPE\_SUITE'].value\_counts()

suite\_val\_y0 = []  
suite\_val\_y1 = []

for val in suite\_val.index:  
 suite\_val\_y1.append(np.sum(application['TARGET']

[application['NAME\_TYPE\_SUITE']==val] == 1))  
 suite\_val\_y0.append(np.sum(application['TARGET']

[application['NAME\_TYPE\_SUITE']==val] == 0))

data = [go.Bar(x = suite\_val.index, y = ((suite\_val\_y1 / suite\_val.sum()) \* 100), name='Yes' ),  
 go.Bar(x = suite\_val.index, y = ((suite\_val\_y0 / suite\_val.sum()) \* 100), name='No' )]

layout = go.Layout(  
 title = "Who accompanied client when applying for the application in terms of loan is repayed or not in %",  
 xaxis=dict(  
 title='Name of type of the Suite',  
 ),  
 yaxis=dict(  
 title='Count of applicants in %',  
 )  
)

fig = go.Figure(data = data, layout=layout)   
fig.layout.template = 'plotly\_dark'

py.iplot(fig)

A screenshot of a cell phone

Description automatically generated

**Distribution of Income sources of Applicants in terms of loan is repayed or not.**

income\_val = application['NAME\_INCOME\_TYPE'].value\_counts()

income\_val\_y0 = []  
income\_val\_y1 = []

for val in income\_val.index:  
 income\_val\_y1.append(np.sum(application['TARGET']

[application['NAME\_INCOME\_TYPE']==val] == 1))  
 income\_val\_y0.append(np.sum(application['TARGET']

[application['NAME\_INCOME\_TYPE']==val] == 0))

data = [go.Bar(x = income\_val.index, y = ((income\_val\_y1 / income\_val.sum()) \* 100), name='Yes' ),  
 go.Bar(x = income\_val.index, y = ((income\_val\_y0 / income\_val.sum()) \* 100), name='No' )]

layout = go.Layout(  
 title = "Income sources of Applicants in terms of loan is repayed or not in %",  
 xaxis=dict(  
 title='Income source',  
 ),  
 yaxis=dict(  
 title='Count of applicants in %',  
 )  
)

fig = go.Figure(data = data, layout=layout)   
fig.layout.template = 'plotly\_dark'

py.iplot(fig)

**A picture containing computer

Description automatically generated**

Observations:

* 1. All the Students and Businessman are repaying loan

**Distribution of Education of Applicants in terms of loan is repayed or not.**

education\_val = application['NAME\_EDUCATION\_TYPE'].value\_counts()

education\_val\_y0 = []  
education\_val\_y1 = []

for val in education\_val.index:  
 education\_val\_y1.append(np.sum(application['TARGET']

[application['NAME\_EDUCATION\_TYPE']==val] == 1))  
 education\_val\_y0.append(np.sum(application['TARGET']

[application['NAME\_EDUCATION\_TYPE']==val] == 0))

data = [go.Bar(x = education\_val.index, y = ((education\_val\_y1 / education\_val.sum()) \* 100), name='Yes' ),  
 go.Bar(x = education\_val.index, y = ((education\_val\_y0 / education\_val.sum()) \* 100), name='No' )]

layout = go.Layout(  
 title = "Education sources of Applicants in terms of loan is repayed or not in %",  
 xaxis=dict(  
 title='Education of Applicants',  
 ),  
 yaxis=dict(  
 title='Count of applicants in %',  
 )  
)

fig = go.Figure(data = data, layout=layout)   
fig.layout.template = 'plotly\_dark'

py.iplot(fig)

**A screenshot of a cell phone

Description automatically generated**

Observations:

1. People with Academic Degree are more likely to repay the loan(Out of 164, only 3 applicants are not able to repay)

**Distribution of Family status of Applicants in terms of loan is repayed or not.**

A picture containing computer

Description automatically generated

Observations:

1. Widows are more likely to repay the loan when compared to appliants with the other family statuses.

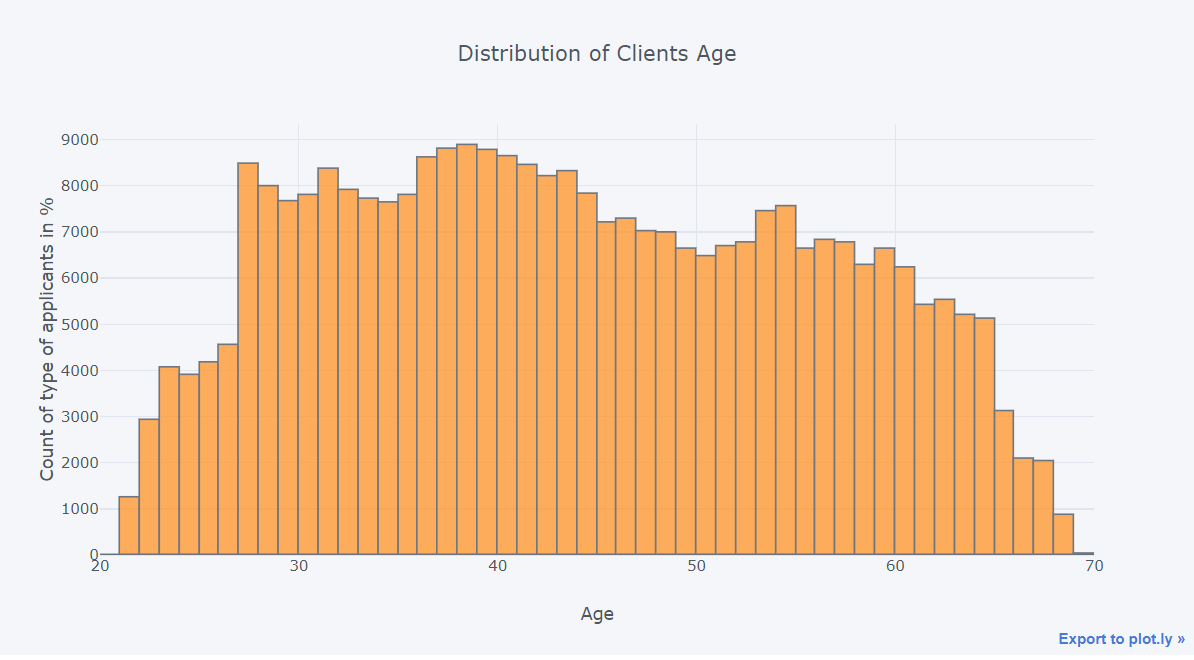
**Distribution of Housing type of Applicants in terms of loan is repayed or not.**

A screenshot of a cell phone

Description automatically generated

**Distribution of Clients Age**

cf.set\_config\_file(theme='pearl')  
(application['DAYS\_BIRTH']/(-365)).iplot(kind='histogram',   
xTitle = 'Age', bins=50,  
yTitle='Count of type of applicants in %',  
title='Distribution of Clients Age')



**Distribution of years before the application the person started current employment.**

cf.set\_config\_file(theme='pearl')  
(application['DAYS\_EMPLOYED']).iplot(kind='histogram',   
xTitle = 'Days',bins=50,  
yTitle='Count of applicants in %',  
title='Days before the application the person started current employment')

A picture containing black, white

Description automatically generated

Observations:

1. The data looks strange(we have -1000.66 years(-365243 days) of employment which is impossible) looks like there is data entry error.

error = application[application['DAYS\_EMPLOYED'] == 365243]  
print('The no of errors are :', len(error))  
(error['TARGET'].value\_counts()/len(error))\*100

A close up of a logo

Description automatically generated

# Create an error flag column  
application['DAYS\_EMPLOYED\_ERROR'] = application["DAYS\_EMPLOYED"] == 365243# Replace the error values with nan  
application['DAYS\_EMPLOYED'].replace({365243: np.nan}, inplace = True)

**Created a seperate column ‘DAYS\_EMPLOYED\_ERROR’, which flags the error.**

cf.set\_config\_file(theme='pearl')  
(application['DAYS\_EMPLOYED']/(-365)).iplot(kind='histogram', xTitle = 'Years of Employment',bins=50,  
yTitle='Count of applicants in %',  
title='Years before the application the person started current employment')

A screenshot of a cell phone

Description automatically generated

application[application['DAYS\_EMPLOYED']>(-365\*2)]['TARGET'].value\_counts()/sum(application['DAYS\_EMPLOYED']>(-365\*2))



Observations:

1. The applicants with less than 2 years of employment are less likely to repay the loan.

**Data Preparation:**

# Flag to represent when Total income is greater than Credit  
application['INCOME\_GT\_CREDIT\_FLAG'] = application['AMT\_INCOME\_TOTAL'] > application['AMT\_CREDIT'] #Column to represent Credit Income Percent

application['CREDIT\_INCOME\_PERCENT'] = application['AMT\_CREDIT'] / application['AMT\_INCOME\_TOTAL']. #Column to represent Annuity Income percent

application['ANNUITY\_INCOME\_PERCENT'] = application['AMT\_ANNUITY'] / application['AMT\_INCOME\_TOTAL'] #Column to represent Credit Term

application['CREDIT\_TERM'] = application['AMT\_CREDIT'] / application['AMT\_ANNUITY']

#Column to represent Days Employed percent in his life

application['DAYS\_EMPLOYED\_PERCENT'] = application['DAYS\_EMPLOYED'] / application['DAYS\_BIRTH'] #Shape of Application data

print('The shape of application data:",application.shape)



**Using Bureau Data:**

print('Reading the data....', end='')  
bureau = pd.read\_csv('bureau.csv')  
print('done!!!')  
print('The shape of data:',bureau.shape)  
print('First 5 rows of data:')  
bureau.head()

A screenshot of a computer screen

Description automatically generated

**Joining Bureau data to Application data:**

# Combining numerical features  
grp = bureau.drop(['SK\_ID\_BUREAU'], axis = 1).groupby(by=['SK\_ID\_CURR']).mean().reset\_index()  
grp.columns = ['BUREAU\_'+column if column !='SK\_ID\_CURR' else column for column in grp.columns]  
application\_bureau = application.merge(grp, on='SK\_ID\_CURR', how='left')  
application\_bureau.update(application\_bureau[grp.columns].fillna(0))

# Combining categorical features  
bureau\_categorical = pd.get\_dummies(bureau.select\_dtypes('object'))  
bureau\_categorical['SK\_ID\_CURR'] = bureau['SK\_ID\_CURR']

grp = bureau\_categorical.groupby(by = ['SK\_ID\_CURR']).mean().reset\_index()  
grp.columns = ['BUREAU\_'+column if column !='SK\_ID\_CURR' else column for column in grp.columns]  
application\_bureau = application\_bureau.merge(grp, on='SK\_ID\_CURR', how='left')  
application\_bureau.update(application\_bureau[grp.columns].fillna(0))

# Shape of application and bureau data combined  
print('The shape application and bureau data combined:',application\_bureau.shape)



**Feature Engineering of Bureau Data:**

# Number of past loans per customer

grp = bureau.groupby(by = ['SK\_ID\_CURR'])['SK\_ID\_BUREAU'].count().reset\_index().rename(columns = {'SK\_ID\_BUREAU': 'BUREAU\_LOAN\_COUNT'})

application\_bureau = application\_bureau.merge(grp, on='SK\_ID\_CURR', how='left')

application\_bureau['BUREAU\_LOAN\_COUNT'] = application\_bureau['BUREAU\_LOAN\_COUNT'].fillna(0)

# Number of types of past loans per customer

grp = bureau[['SK\_ID\_CURR', 'CREDIT\_TYPE']].groupby(by = ['SK\_ID\_CURR'])['CREDIT\_TYPE'].nunique().reset\_index().rename(columns={'CREDIT\_TYPE': 'BUREAU\_LOAN\_TYPES'})

application\_bureau = application\_bureau.merge(grp, on='SK\_ID\_CURR', how='left')

application\_bureau['BUREAU\_LOAN\_TYPES'] = application\_bureau['BUREAU\_LOAN\_TYPES'].fillna(0)

# Debt over credit ratio

bureau['AMT\_CREDIT\_SUM'] = bureau['AMT\_CREDIT\_SUM'].fillna(0)

bureau['AMT\_CREDIT\_SUM\_DEBT'] = bureau['AMT\_CREDIT\_SUM\_DEBT'].fillna(0)

grp1 = bureau[['SK\_ID\_CURR','AMT\_CREDIT\_SUM']].groupby(by=['SK\_ID\_CURR'])['AMT\_CREDIT\_SUM'].sum().reset\_index().rename(columns={'AMT\_CREDIT\_SUM': 'TOTAL\_CREDIT\_SUM'})

grp2 = bureau[['SK\_ID\_CURR','AMT\_CREDIT\_SUM\_DEBT']].groupby(by=['SK\_ID\_CURR'])['AMT\_CREDIT\_SUM\_DEBT'].sum().reset\_index().rename(columns={'AMT\_CREDIT\_SUM\_DEBT':'TOTAL\_CREDIT\_SUM\_DEBT'})

grp1['DEBT\_CREDIT\_RATIO'] = grp2['TOTAL\_CREDIT\_SUM\_DEBT']/grp1['TOTAL\_CREDIT\_SUM']

del grp1['TOTAL\_CREDIT\_SUM']

application\_bureau = application\_bureau.merge(grp1, on='SK\_ID\_CURR', how='left')

application\_bureau['DEBT\_CREDIT\_RATIO'] = application\_bureau['DEBT\_CREDIT\_RATIO'].fillna(0)

application\_bureau['DEBT\_CREDIT\_RATIO'] = application\_bureau.replace([np.inf, -np.inf], 0)

application\_bureau['DEBT\_CREDIT\_RATIO'] = pd.to\_numeric(application\_bureau['DEBT\_CREDIT\_RATIO'], downcast='float')

# Overdue over debt ratio

bureau['AMT\_CREDIT\_SUM\_OVERDUE'] = bureau['AMT\_CREDIT\_SUM\_OVERDUE'].fillna(0)

bureau['AMT\_CREDIT\_SUM\_DEBT'] = bureau['AMT\_CREDIT\_SUM\_DEBT'].fillna(0)

grp1 = bureau[['SK\_ID\_CURR','AMT\_CREDIT\_SUM\_OVERDUE']].groupby(by=['SK\_ID\_CURR'])['AMT\_CREDIT\_SUM\_OVERDUE'].sum().reset\_index().rename(columns={'AMT\_CREDIT\_SUM\_OVERDUE': 'TOTAL\_CUSTOMER\_OVERDUE'})

grp2 = bureau[['SK\_ID\_CURR','AMT\_CREDIT\_SUM\_DEBT']].groupby(by=['SK\_ID\_CURR'])['AMT\_CREDIT\_SUM\_DEBT'].sum().reset\_index().rename(columns={'AMT\_CREDIT\_SUM\_DEBT':'TOTAL\_CUSTOMER\_DEBT'})

grp1['OVERDUE\_DEBT\_RATIO'] = grp1['TOTAL\_CUSTOMER\_OVERDUE']/grp2['TOTAL\_CUSTOMER\_DEBT']

del grp1['TOTAL\_CUSTOMER\_OVERDUE']

application\_bureau = application\_bureau.merge(grp1, on='SK\_ID\_CURR', how='left')

application\_bureau['OVERDUE\_DEBT\_RATIO'] = application\_bureau['OVERDUE\_DEBT\_RATIO'].fillna(0)

application\_bureau['OVERDUE\_DEBT\_RATIO'] = application\_bureau.replace([np.inf, -np.inf], 0)

application\_bureau['OVERDUE\_DEBT\_RATIO'] = pd.to\_numeric(application\_bureau['OVERDUE\_DEBT\_RATIO'], downcast='float')

**Using Previous Application Data:**

print('Reading the data....', end='')  
previous\_applicaton = pd.read\_csv('previous\_application.csv')  
print('done!!!')  
print('The shape of data:',previous\_applicaton.shape)  
print('First 5 rows of data:')  
previous\_applicaton.head()

A screenshot of a cell phone

Description automatically generated

**Joining Previous Application data to Application Bureau data:**

# Number of previous applications per customer

grp = previous\_applicaton[['SK\_ID\_CURR','SK\_ID\_PREV']].groupby(by=['SK\_ID\_CURR'])['SK\_ID\_PREV'].count().reset\_index().rename(columns={'SK\_ID\_PREV':'PREV\_APP\_COUNT'})

application\_bureau\_prev = application\_bureau.merge(grp, on =['SK\_ID\_CURR'], how = 'left')

application\_bureau\_prev['PREV\_APP\_COUNT'] = application\_bureau\_prev['PREV\_APP\_COUNT'].fillna(0)

# Combining numerical features

grp = previous\_applicaton.drop('SK\_ID\_PREV', axis =1).groupby(by=['SK\_ID\_CURR']).mean().reset\_index()

prev\_columns = ['PREV\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns ]

grp.columns = prev\_columns

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on =['SK\_ID\_CURR'], how = 'left')

application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

# Combining categorical features

prev\_categorical = pd.get\_dummies(previous\_applicaton.select\_dtypes('object'))

prev\_categorical['SK\_ID\_CURR'] = previous\_applicaton['SK\_ID\_CURR']

prev\_categorical.head()

grp = prev\_categorical.groupby('SK\_ID\_CURR').mean().reset\_index()

grp.columns = ['PREV\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns]

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on=['SK\_ID\_CURR'], how='left')

application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

**Using POS\_CASH\_balance data:**

print('Reading the data....', end='')  
pos\_cash = pd.read\_csv('POS\_CASH\_balance.csv')  
print('done!!!')  
print('The shape of data:',pos\_cash.shape)  
print('First 5 rows of data:')  
pos\_cash.head()

A screenshot of a cell phone

Description automatically generated

**Joining POS\_CASH\_balance data to application\_bureau\_prev\_data:**

# Combining numerical features

grp = pos\_cash.drop('SK\_ID\_PREV', axis =1).groupby(by=['SK\_ID\_CURR']).mean().reset\_index()

prev\_columns = ['POS\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns ]

grp.columns = prev\_columns

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on =['SK\_ID\_CURR'], how = 'left')

application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

# Combining categorical features

pos\_cash\_categorical = pd.get\_dummies(pos\_cash.select\_dtypes('object'))

pos\_cash\_categorical['SK\_ID\_CURR'] = pos\_cash['SK\_ID\_CURR']

grp = pos\_cash\_categorical.groupby('SK\_ID\_CURR').mean().reset\_index()

grp.columns = ['POS\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns]

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on=['SK\_ID\_CURR'], how='left')

application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

**Using installments\_payments data:**

print('Reading the data....', end='')  
insta\_payments = pd.read\_csv('installments\_payments.csv')  
print('done!!!')  
print('The shape of data:',insta\_payments.shape)  
print('First 5 rows of data:')  
insta\_payments.head()

A screenshot of a social media post

Description automatically generated

**Joining Installments Payments data to application\_bureau\_prev\_data:**

# Combining numerical features and there are no categorical features in this dataset  
grp = insta\_payments.drop('SK\_ID\_PREV', axis =1).groupby(by=['SK\_ID\_CURR']).mean().reset\_index()

prev\_columns = ['INSTA\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns ]

grp.columns = prev\_columns

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on =['SK\_ID\_CURR'], how = 'left')  
application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

**Using Credit card balance data:**

print('Reading the data....', end='')  
credit\_card = pd.read\_csv('credit\_card\_balance.csv')  
print('done!!!')  
print('The shape of data:',credit\_card.shape)  
print('First 5 rows of data:')  
credit\_card.head()

A screenshot of a social media post

Description automatically generated

**Joining Credit card balance data to application\_bureau\_prev data:**

# Combining numerical features

grp = credit\_card.drop('SK\_ID\_PREV', axis =1).groupby(by=['SK\_ID\_CURR']).mean().reset\_index()

prev\_columns = ['CREDIT\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns ]

grp.columns = prev\_columns

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on =['SK\_ID\_CURR'], how = 'left')

application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

# Combining categorical features

credit\_categorical = pd.get\_dummies(credit\_card.select\_dtypes('object'))

credit\_categorical['SK\_ID\_CURR'] = credit\_card['SK\_ID\_CURR']

grp = credit\_categorical.groupby('SK\_ID\_CURR').mean().reset\_index()

grp.columns = ['CREDIT\_'+column if column != 'SK\_ID\_CURR' else column for column in grp.columns]

application\_bureau\_prev = application\_bureau\_prev.merge(grp, on=['SK\_ID\_CURR'], how='left')

application\_bureau\_prev.update(application\_bureau\_prev[grp.columns].fillna(0))

Shape of final prepared data: (307511, 377)

**Dividing final data into train, valid and test datasets:**

y = application\_bureau\_prev.pop('TARGET').valuesX\_train, X\_temp, y\_train, y\_temp = train\_test\_split(application\_bureau\_prev.drop(['SK\_ID\_CURR'],axis=1), y, stratify = y, test\_size=0.3, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, stratify = y\_temp, test\_size=0.5, random\_state=42)print('Shape of X\_train:',X\_train.shape)

print('Shape of X\_val:',X\_val.shape)  
print('Shape of X\_test:',X\_test.shape)



**Featurizing the data:**

#Seperation of columns into numeric and categorical columns

types = np.array([dt for dt in X\_train.dtypes])

all\_columns = X\_train.columns.values

is\_num = types != 'object'

num\_cols = all\_columns[is\_num]

cat\_cols = all\_columns[~is\_num]

#Featurization of numeric data

imputer\_num = SimpleImputer(strategy='median')

X\_train\_num = imputer\_num.fit\_transform(X\_train[num\_cols])

X\_val\_num = imputer\_num.transform(X\_val[num\_cols])

X\_test\_num = imputer\_num.transform(X\_test[num\_cols])

scaler\_num = StandardScaler()

X\_train\_num1 = scaler\_num.fit\_transform(X\_train\_num)

X\_val\_num1 = scaler\_num.transform(X\_val\_num)

X\_test\_num1 = scaler\_num.transform(X\_test\_num)

X\_train\_num\_final = pd.DataFrame(X\_train\_num1, columns=num\_cols)

X\_val\_num\_final = pd.DataFrame(X\_val\_num1, columns=num\_cols)

X\_test\_num\_final = pd.DataFrame(X\_test\_num1, columns=num\_cols)

# Featurization of categorical data

imputer\_cat = SimpleImputer(strategy='constant', fill\_value='MISSING')

X\_train\_cat = imputer\_cat.fit\_transform(X\_train[cat\_cols])

X\_val\_cat = imputer\_cat.transform(X\_val[cat\_cols])

X\_test\_cat = imputer\_cat.transform(X\_test[cat\_cols])

X\_train\_cat1= pd.DataFrame(X\_train\_cat, columns=cat\_cols)

X\_val\_cat1= pd.DataFrame(X\_val\_cat, columns=cat\_cols)

X\_test\_cat1= pd.DataFrame(X\_test\_cat, columns=cat\_cols)

ohe = OneHotEncoder(sparse=False,handle\_unknown='ignore')

X\_train\_cat2 = ohe.fit\_transform(X\_train\_cat1)

X\_val\_cat2 = ohe.transform(X\_val\_cat1)

X\_test\_cat2 = ohe.transform(X\_test\_cat1)

cat\_cols\_ohe = list(ohe.get\_feature\_names(input\_features=cat\_cols))

X\_train\_cat\_final = pd.DataFrame(X\_train\_cat2, columns = cat\_cols\_ohe)

X\_val\_cat\_final = pd.DataFrame(X\_val\_cat2, columns = cat\_cols\_ohe)

X\_test\_cat\_final = pd.DataFrame(X\_test\_cat2, columns = cat\_cols\_ohe)

# Final complete data

X\_train\_final = pd.concat([X\_train\_num\_final,X\_train\_cat\_final], axis = 1)

X\_val\_final = pd.concat([X\_val\_num\_final,X\_val\_cat\_final], axis = 1)

X\_test\_final = pd.concat([X\_test\_num\_final,X\_test\_cat\_final], axis = 1)

print(X\_train\_final.shape)

print(X\_val\_final.shape)

print(X\_test\_final.shape)



**Saving the files for future use:**

# Saving the Dataframes into CSV files for future use

X\_train\_final.to\_csv('X\_train\_final.csv')

X\_val\_final.to\_csv('X\_val\_final.csv')

X\_test\_final.to\_csv('X\_test\_final.csv')

# Saving the numpy arrays into text files for future use

np.savetxt('y.txt', y)

np.savetxt('y\_train.txt', y\_train)

np.savetxt('y\_val.txt', y\_val)

np.savetxt('y\_test.txt', y\_test)

## ****Selection of features:****

model\_sk = lgb.LGBMClassifier(boosting\_type='gbdt', max\_depth=7, learning\_rate=0.01, n\_estimators= 2000,

class\_weight='balanced', subsample=0.9, colsample\_bytree= 0.8, n\_jobs=-1)

train\_features, valid\_features, train\_y, valid\_y = train\_test\_split(X\_train\_final, y\_train, test\_size = 0.15, random\_state = 42)

model\_sk.fit(train\_features, train\_y, early\_stopping\_rounds=100, eval\_set = [(valid\_features, valid\_y)], eval\_metric = 'auc', verbose = 200)

A screenshot of a cell phone

Description automatically generated

**Training LGBM**

feature\_imp = pd.DataFrame(sorted(zip(model\_sk.feature\_importances\_, X\_train\_final.columns)), columns=['Value','Feature'])  
features\_df = feature\_imp.sort\_values(by="Value", ascending=False)  
selected\_features = list(features\_df[features\_df['Value']>=50]['Feature'])

# Saving the selected features into pickle file  
with open('select\_features.txt','wb') as fp:  
 pickle.dump(selected\_features, fp)

print('The no. of features selected:',len(selected\_features))



# Feature importance Plot  
data1 = features\_df.head(20)  
data = [go.Bar(x =data1.sort\_values(by='Value')['Value'] , y = data1.sort\_values(by='Value')['Feature'], orientation = 'h',  
 marker = dict(  
 color = 'rgba(43, 13, 150, 0.6)',  
 line = dict(  
 color = 'rgba(43, 13, 150, 1.0)',  
 width = 1.5)  
 )) ]

layout = go.Layout(  
 autosize=False,  
 width=1300,  
 height=700,  
 title = "Top 20 important features",  
 xaxis=dict(  
 title='Importance value'  
 ),  
 yaxis=dict(  
 automargin=True  
 ),  
 bargap=0.4  
 )

fig = go.Figure(data = data, layout=layout)  
fig.layout.template = 'seaborn'

py.iplot(fig)

A screenshot of a cell phone

Description automatically generated

# **Machine Learning Models:**

I have tried Logistic **Regression, Random Forest and LightGBM machine** learning models.

Reusable functions for plotting Confusion matrix and CV plot.

def plot\_confusion\_matrix(test\_y, predicted\_y):

# Confusion matrix

C = confusion\_matrix(test\_y, predicted\_y)

# Recall matrix

A = (((C.T)/(C.sum(axis=1))).T)

# Precision matrix

B = (C/C.sum(axis=0))

plt.figure(figsize=(20,4))

labels = ['Re-paid(0)','Not Re-paid(1)']

cmap=sns.light\_palette("purple")

plt.subplot(1,3,1)

sns.heatmap(C, annot=True, cmap=cmap,fmt="d", xticklabels = labels, yticklabels=labels)

plt.xlabel('Predicted Class')

plt.ylabel('Orignal Class')

plt.title('Confusion matrix')

plt.subplot(1,3,2)

sns.heatmap(A, annot=True, cmap=cmap, xticklabels = labels, yticklabels=labels)

plt.xlabel('Predicted Class')

plt.ylabel('Orignal Class')

plt.title('Recall matrix')

plt.subplot(1,3,3)

sns.heatmap(B, annot=True, cmap=cmap, xticklabels = labels, yticklabels=labels)

plt.xlabel('Predicted Class')

plt.ylabel('Orignal Class')

plt.title('Precision matrix')

plt.show()

def cv\_plot(alpha, cv\_auc):

fig, ax = plt.subplots()

ax.plot(np.log10(alpha), cv\_auc,c='g')

for i, txt in enumerate(np.round(cv\_auc,3)):

ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv\_auc[i]))

plt.grid()

plt.xticks(np.log10(alpha))

plt.title("Cross Validation Error for each alpha")

plt.xlabel("Alpha i's")

plt.ylabel("Error measure")

plt.show()

**Logistic regression with selected features:**

Logistic Regression finds a hyperplane which best seperates the given positive and negative data points.

alpha = np.logspace(-4,4,9)  
cv\_auc\_score = []

for i in alpha:  
 clf = SGDClassifier(alpha=i, penalty='l1',class\_weight = 'balanced', loss='log', random\_state=28)  
 clf.fit(X\_train\_final[selected\_features], y\_train)  
 sig\_clf = CalibratedClassifierCV(clf, method='sigmoid')  
 sig\_clf.fit(X\_train\_final[selected\_features], y\_train)  
 y\_pred\_prob = sig\_clf.predict\_proba(X\_val\_final[selected\_features])[:,1]  
 cv\_auc\_score.append(roc\_auc\_score(y\_val,y\_pred\_prob))  
 print('For alpha {0}, cross validation AUC score {1}'.format(i,roc\_auc\_score(y\_val,y\_pred\_prob)))

cv\_plot(alpha, cv\_auc\_score)

print('The Optimal C value is:', alpha[np.argmax(cv\_auc\_score)])

A close up of a map

Description automatically generated

Cross validation results and plot for Logistic Regression model.

best\_alpha = alpha[np.argmax(cv\_auc\_score)]

logreg = SGDClassifier(alpha = best\_alpha, class\_weight = 'balanced', penalty = 'l1', loss='log', random\_state = 28)

logreg.fit(X\_train\_final[selected\_features], y\_train)

logreg\_sig\_clf = CalibratedClassifierCV(logreg, method='sigmoid')

logreg\_sig\_clf.fit(X\_train\_final[selected\_features], y\_train)

y\_pred\_prob = logreg\_sig\_clf.predict\_proba(X\_train\_final[selected\_features])[:,1]

print('For best alpha {0}, The Train AUC score is {1}'.format(best\_alpha, roc\_auc\_score(y\_train,y\_pred\_prob) ))

y\_pred\_prob = logreg\_sig\_clf.predict\_proba(X\_val\_final[selected\_features])[:,1]

print('For best alpha {0}, The Cross validated AUC score is {1}'.format(best\_alpha, roc\_auc\_score(y\_val,y\_pred\_prob) ))

y\_pred\_prob = logreg\_sig\_clf.predict\_proba(X\_test\_final[selected\_features])[:,1]

print('For best alpha {0}, The Test AUC score is {1}'.format(best\_alpha, roc\_auc\_score(y\_test,y\_pred\_prob) ))

y\_pred = logreg.predict(X\_test\_final[selected\_features])

print('The test AUC score is :', roc\_auc\_score(y\_test,y\_pred\_prob))

print('The percentage of misclassified points {:05.2f}% :'.format((1-accuracy\_score(y\_test, y\_pred))\*100))

plot\_confusion\_matrix(y\_test, y\_pred)

A screenshot of a cell phone

Description automatically generated

Logistic Regression model results

from sklearn.metrics import roc\_curve  
fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)  
auc = roc\_auc\_score(y\_test,y\_pred\_prob)plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, marker='.')  
plt.plot([0, 1], [0, 1], linestyle='--')  
plt.title('ROC curve', fontsize = 20)  
plt.xlabel('FPR', fontsize=15)  
plt.ylabel('TPR', fontsize=15)  
plt.grid()

plt.legend(["AUC=%.3f"%auc])

plt.show()

A close up of a map

Description automatically generated

ROC curve for Logistic Regression model with AUC=0.754

**Random Forest with selected features:**

The Random Forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a “forest”), this model uses two key concepts that gives it the name random:

1. Random sampling of training data points when building trees
2. Random subsets of features considered when splitting nodes

alpha = [200,500,1000,2000]  
max\_depth = [7, 10]  
cv\_auc\_score = []

for i in alpha:

for j in max\_depth:

clf = RandomForestClassifier(n\_estimators=i, criterion='gini', max\_depth=j,class\_weight='balanced',  
random\_state=42, n\_jobs=-1)  
clf.fit(X\_train\_final[selected\_features], y\_train)  
sig\_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig\_clf.fit(X\_train\_final[selected\_features], y\_train)  
y\_pred\_prob = sig\_clf.predict\_proba(X\_val\_final[selected\_features])[:,1]  
cv\_auc\_score.append(roc\_auc\_score(y\_val,y\_pred\_prob))  
print('For n\_estimators {0}, max\_depth {1} cross validation AUC score {2}'.  
format(i,j,roc\_auc\_score(y\_val,y\_pred\_prob)))

A screenshot of a cell phone

Description automatically generated

Cross validation results for Random Forest model.

best\_alpha = np.argmax(cv\_auc\_score)

print('The optimal values are: n\_estimators {0}, max\_depth {1} '.format(alpha[int(best\_alpha/2)], max\_depth[int(best\_alpha%2)]))

rf = RandomForestClassifier(n\_estimators=alpha[int(best\_alpha/2)], criterion='gini', max\_depth=max\_depth[int(best\_alpha%2)],

class\_weight='balanced', random\_state=42, n\_jobs=-1)

rf.fit(X\_train\_final[selected\_features], y\_train)

rf\_sig\_clf = CalibratedClassifierCV(rf, method="sigmoid")

rf\_sig\_clf.fit(X\_train\_final[selected\_features], y\_train)

y\_pred\_prob = rf\_sig\_clf.predict\_proba(X\_train\_final[selected\_features])[:,1]

print('For best n\_estimators {0} best max\_depth {1}, The Train AUC score is {2}'.format(alpha[int(best\_alpha/2)],

max\_depth[int(best\_alpha%2)],roc\_auc\_score(y\_train,y\_pred\_prob)))

y\_pred\_prob = rf\_sig\_clf.predict\_proba(X\_val\_final[selected\_features])[:,1]

print('For best n\_estimators {0} best max\_depth {1}, The Validation AUC score is {2}'.format(alpha[int(best\_alpha/2)],

max\_depth[int(best\_alpha%2)],roc\_auc\_score(y\_val,y\_pred\_prob)))

y\_pred\_prob = rf\_sig\_clf.predict\_proba(X\_test\_final[selected\_features])[:,1]

print('For best n\_estimators {0} best max\_depth {1}, The Test AUC score is {2}'.format(alpha[int(best\_alpha/2)],

max\_depth[int(best\_alpha%2)],roc\_auc\_score(y\_test,y\_pred\_prob)))

y\_pred = rf\_sig\_clf.predict(X\_test\_final[selected\_features])

print('The test AUC score is :', roc\_auc\_score(y\_test,y\_pred\_prob))

print('The percentage of misclassified points {:05.2f}% :'.format((1-accuracy\_score(y\_test, y\_pred))\*100))

plot\_confusion\_matrix(y\_test, y\_pred)

A screenshot of a cell phone

Description automatically generated

Random Forest model results.

from sklearn.metrics import roc\_curve  
fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

auc = roc\_auc\_score(y\_test,y\_pred\_prob)plt.figure(figsize=(8,6))  
plt.plot(fpr, tpr, marker='.')  
plt.plot([0, 1], [0, 1], linestyle='--')  
plt.title('ROC curve', fontsize = 20)  
plt.xlabel('FPR', fontsize=15)  
plt.ylabel('TPR', fontsize=15)  
plt.grid()  
plt.legend(["AUC=%.3f"%auc])  
plt.show()

A close up of a map

Description automatically generated

ROC curve for Random Forest model with AUC=0.75

**LightGBM with selected features:**

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

weight = np.ones((len(X\_train\_final),), dtype=int)  
for i in range(len(X\_train\_final)):  
 if y\_train[i]== 0:  
 weight[i]=1  
 else:  
 weight[i]=11  
train\_data=lgb.Dataset(X\_train\_final[selected\_features], label = y\_train, weight= weight )  
valid\_data=lgb.Dataset(X\_val\_final[selected\_features], label = y\_val)cv\_auc\_score = []  
max\_depth = [3, 5, 7, 10]for i in max\_depth:  
   
 params = {'boosting\_type': 'gbdt',  
 'max\_depth' : i,  
 'objective': 'binary',  
 'nthread': 5,  
 'num\_leaves': 32,  
 'learning\_rate': 0.05,  
 'max\_bin': 512,  
 'subsample\_for\_bin': 200,  
 'subsample': 0.7,  
 'subsample\_freq': 1,  
 'colsample\_bytree': 0.8,  
 'reg\_alpha': 20,  
 'reg\_lambda': 20,  
 'min\_split\_gain': 0.5,  
 'min\_child\_weight': 1,  
 'min\_child\_samples': 10,  
 'scale\_pos\_weight': 1,  
 'num\_class' : 1,  
 'metric' : 'auc'  
 }

lgbm = lgb.train(params,  
 train\_data,  
 2500,  
 valid\_sets=valid\_data,  
 early\_stopping\_rounds= 100,  
 verbose\_eval= 10  
 )  
 y\_pred\_prob = lgbm.predict(X\_val\_final[selected\_features])  
 cv\_auc\_score.append(roc\_auc\_score(y\_val,y\_pred\_prob))  
 print('For max\_depth {0} and some other parameters, cross validation AUC score {1}'.format(i,roc\_auc\_score(y\_val,y\_pred\_prob)))

print('The optimal max\_depth: ', max\_depth[np.argmax(cv\_auc\_score)])

params = {'boosting\_type': 'gbdt',  
 'max\_depth' : max\_depth[np.argmax(cv\_auc\_score)],  
 'objective': 'binary',  
 'nthread': 5,  
 'num\_leaves': 32,  
 'learning\_rate': 0.05,  
 'max\_bin': 512,  
 'subsample\_for\_bin': 200,  
 'subsample': 0.7,  
 'subsample\_freq': 1,  
 'colsample\_bytree': 0.8,  
 'reg\_alpha': 20,  
 'reg\_lambda': 20,  
 'min\_split\_gain': 0.5,  
 'min\_child\_weight': 1,  
 'min\_child\_samples': 10,  
 'scale\_pos\_weight': 1,  
 'num\_class' : 1,  
 'metric' : 'auc'  
 }

lgbm = lgb.train(params,  
 train\_data,  
 2500,  
 valid\_sets=valid\_data,  
 early\_stopping\_rounds= 100,  
 verbose\_eval= 10  
 )

y\_pred\_prob = lgbm.predict(X\_train\_final[selected\_features])  
print('For best max\_depth {0}, The Train AUC score is {1}'.format(max\_depth[np.argmax(cv\_auc\_score)], roc\_auc\_score(y\_train,y\_pred\_prob) ))

y\_pred\_prob = lgbm.predict(X\_val\_final[selected\_features])  
print('For best max\_depth {0}, The Cross validated AUC score is {1}'.format(max\_depth[np.argmax(cv\_auc\_score)], roc\_auc\_score(y\_val,y\_pred\_prob) ))

y\_pred\_prob = lgbm.predict(X\_test\_final[selected\_features])  
print('For best max\_depth {0}, The Test AUC score is {1}'.format(max\_depth[np.argmax(cv\_auc\_score)],

roc\_auc\_score(y\_test,y\_pred\_prob) ))

y\_pred = np.ones((len(X\_test\_final),), dtype=int)

for i in range(len(y\_pred\_prob)):  
 if y\_pred\_prob[i]<=0.5:  
 y\_pred[i]=0  
 else:  
 y\_pred[i]=1

print('The test AUC score is :', roc\_auc\_score(y\_test,y\_pred\_prob))

print('The percentage of misclassified points {:05.2f}% :'.format((1-accuracy\_score(y\_test, y\_pred))\*100))

plot\_confusion\_matrix(y\_test, y\_pred)

A screenshot of a cell phone

Description automatically generated

LightGBM model Results

from sklearn.metrics import roc\_curve  
fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)  
auc = roc\_auc\_score(y\_test,y\_pred\_prob)plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, marker='.')  
plt.plot([0, 1], [0, 1], linestyle='--')  
plt.title('ROC curve', fontsize = 20)  
plt.xlabel('FPR', fontsize=15)  
plt.ylabel('TPR', fontsize=15)  
plt.grid()  
plt.legend(["AUC=%.3f"%auc])  
plt.show()

A close up of a map

Description automatically generated

**ROC curve for LightGBM model with AUC=0.787**

## Overview of Results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Train AUC** | **Validation AUC** | **Test AUC** |
| Logistic Regression with Selected features | 0.756 | 0.747 | 0.753 |
| Random Forest with Selected features | 0.841 | 0.751 | 0.751 |
| **LightGBM with Selected features** | **0.861** | **0.781** | **0.787** |

**LightGBM** gives the best performance and it is also faster to train when compared to Xgboost.