```
'''DELINQUENCY:It arises when an activity does not occur at it's scheduled.'''
```

```
'''USE CASE: 1).Mobile financial services (MFS) is more convenient and
efficient, and less costly, than the traditional high-touch model for delivering
 microfinance services
2).Microfinance represents $70 billion in outstanding loans and a global
outreach of 200 million clients.
3).The Consumer is believed to be delinquent if he deviates from the path of
paying back the loaned amount within 5 days
'''
```

```
#Exercise to do:Create a delinquency model which can predict in terms of a
#probability for each loan transaction, whether the customer will be paying
#back the loaned amount within 5 days of insurance of loan
#(Label ‘1’ & ’0’)
#Find Enclosed the Data Description File and The Sample Data for the Modeling Exercise.
```

```
cd="https://docs.google.com/document/d/1ux4chYStCMQTmTeTPJKCKQzRmsQLqAZ6/edit?usp=sharing&ouid=106841864929688907405&r
cd
```

```
!pip install catboost
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/b2/aa/e61819d04ef2bbee778bf4b3a748db1f3ad23512377e43ecfdc32
    |████████████████████████████████| 64.8MB 49kB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from catboost) (1.4.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catboost) (1.12.0)
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from catboost) (0.10.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.0.5)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from catboost) (4.4.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.18.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from catboost) (3.2.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24.0->catbc
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.2
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from plotly->catboost)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catb
Installing collected packages: catboost
Successfully installed catboost-0.23.2
```

```
'''
numpy stands for numerical python and it is used to perform a wide variety of
mathematical operations on arrays
Pandas is mainly used for data analysis. Pandas allows importing data from
 various file formats such as comma-separated
 values, JSON, SQL database tables or queries, and Microsoft Excel
 '''
#random() function generate random floating numbers between 0 and 1.
#Parameters : This method does not accept any parameter. Returns : This method
#returns a random floating number between 0 and 1.
#Seaborn is an open-source Python library built on top of matplotlib. It is used
# for data visualization and exploratory data analysis. Seaborn works easily
#with dataframes and the Pandas library. The graphs created can also be
# customized easily.
#Matplotlib is a comprehensive library for creating static, animated, and
#interactive visualizations in Python. Matplotlib makes easy things easy and
#hard things possible.
#Pickle in Python is primarily used in serializing and deserializing a Python
```

```
# object structure. In other words, it's the process of converting a Python
#object into a byte stream to store it in a file/database, maintain program
#state across sessions, or transport data over the network.
#%matplotlib inline sets the backend of matplotlib to the 'inline' backend: With
#this backend, the output of plotting commands is displayed inline within
#frontends like the Jupyter notebook, directly below the code cell that produced
#it. The resulting plots will then also be stored in the notebook document.
#os.name: The name of the operating system dependent module imported. The
#following names have currently been registered: 'posix', 'nt', 'java'. platform.
#system(): Return the name of the OS system is running on. platform
#Model_selection is a method for setting a blueprint to analyze data and then
#using it to measure new data. Selecting a proper model allows you to generate
#accurate results when making a prediction. To do that, you need to train your
#model by using a specific dataset. Then, you test the model against another
#dataset
#When you do: from datetime import datetime import datetime. You are first
#setting datetime to be a reference to the class, then immediately setting it to
#be a reference to the module. When you do it the other way around, it's the same
#thing, but it ends up being a reference to the class.
'''
#By definition a confusion matrix is such that C i , j is equal to the number of
observations known to be in group and predicted to be in group . Thus in binary
classification, the count of true negatives is C 0 , 0 , false negatives is
C 1 , 0 , true positives is C 1 , 1 and false positives is C 0 , 1 .
#StandardScaler standardizes a feature by subtracting the mean and then scaling
 to unit variance. Unit variance means dividing all the values by the standard
 deviation. StandardScaler does not meet the strict definition of scale I
 introduced earlier.
#Decision Trees (DTs) are a non-parametric supervised learning method used for
classification and regression. The goal is to create a model that predicts the
value of a target variable by learning simple decision rules inferred from the
data features.
#The sklearn. metrics module implements several loss, score, and utility
```

```
functions to measure classification performance. Some metrics might require
probability estimates of the positive class, confidence values, or binary
decisions values.
#train_test_split is a function in Sklearn model selection for splitting data
arrays into two subsets: for training data and for testing data. With this
 function, you don't need to divide the dataset manually. By default,
 Sklearn train_test_split will make random partitions for the two subsets
#datasets import load_iris >>> from sklearn. model_selection import
cross_val_score >>> from sklearn. tree import DecisionTreeClassifier >>>
clf = DecisionTreeClassifier(random_state=0) >>> iris = load_iris() >>>
cross_val_score(clf, iris. data, iris.
'''
import numpy as np
import pandas as pd
import random
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
%matplotlib inline
sns.set(color_codes=True)
import os
from sklearn.model_selection import GridSearchCV
from datetime import datetime
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from sklearn.preprocessing import StandardScaler

from sklearn import tree
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
# Boosting Algorithms :
from xgboost  import XGBClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix, normalized_mutual_info_score
from sklearn.linear_model import SGDClassifier
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.metrics.class
  warnings.warn(message, FutureWarning)
```

```
#train is variable name
#pd.read_csv is used for reading file
train = pd.read_csv('sample_data_intw.csv')
```

Exploratory Data Analysis and Data Preprocessing

```
#train.head() is used for reading first five entries
train.head()
```

```
#train.drop is used for dropping
train.drop('Unnamed: 0',axis=1,inplace=True)
```

```
print("Size of Train data = {}".format(train.shape))
```

```
Size of Train data = (209593, 36)
```

Checks for Null values

```
train.info()
#it is used to find datatype
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 36 columns):
```

```
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   label                 209593 non-null  int64  
 1   msisdn                209593 non-null  object 
 2   aon                   209593 non-null  float64
 3   daily_decr30          209593 non-null  float64
 4   daily_decr90          209593 non-null  float64
 5   rental30              209593 non-null  float64
 6   rental90              209593 non-null  float64
 7   last_rech_date_ma     209593 non-null  float64
 8   last_rech_date_da     209593 non-null  float64
 9   last_rech_amt_ma      209593 non-null  int64  
 10  cnt_ma_rech30         209593 non-null  int64  
 11  fr_ma_rech30          209593 non-null  float64
 12  sumamnt_ma_rech30     209593 non-null  float64
 13  medianamnt_ma_rech30  209593 non-null  float64
 14  medianmarechprebal30  209593 non-null  float64
 15  cnt_ma_rech90         209593 non-null  int64  
 16  fr_ma_rech90          209593 non-null  int64  
 17  sumamnt_ma_rech90     209593 non-null  int64  
 18  medianamnt_ma_rech90  209593 non-null  float64
 19  medianmarechprebal90  209593 non-null  float64
 20  cnt_da_rech30         209593 non-null  float64
 21  fr_da_rech30          209593 non-null  float64
 22  cnt_da_rech90         209593 non-null  int64  
 23  fr_da_rech90          209593 non-null  int64  
 24  cnt_loans30           209593 non-null  int64  
 25  amnt_loans30          209593 non-null  int64  
 26  maxamnt_loans30       209593 non-null  float64
 27  medianamnt_loans30    209593 non-null  float64
 28  cnt_loans90           209593 non-null  float64
 29  amnt_loans90          209593 non-null  int64  
 30  maxamnt_loans90       209593 non-null  int64  
 31  medianamnt_loans90    209593 non-null  float64
 32  payback30             209593 non-null  float64
 33  payback90             209593 non-null  float64
```

```
 34  pcircle               209593 non-null  object
 35  pdate                 209593 non-null  object
dtypes: float64(21), int64(12), object(3)
memory usage: 57.6+ MB
```

```
train.isnull().sum()
#isnull(). sum(). sum() returns the number of missing values in the data set.
#A simple way to deal with data containing missing values is to skip rows with
#missing values in the dataset
```

```
label                   0
msisdn                  0
aon                     0
daily_decr30            0
daily_decr90            0
rental30                0
rental90                0
last_rech_date_ma       0
last_rech_date_da       0
last_rech_amt_ma        0
cnt_ma_rech30           0
fr_ma_rech30            0
sumamnt_ma_rech30       0
medianamnt_ma_rech30    0
medianmarechprebal30    0
cnt_ma_rech90           0
fr_ma_rech90            0
sumamnt_ma_rech90       0
medianamnt_ma_rech90    0
medianmarechprebal90    0
cnt_da_rech30           0
fr_da_rech30            0
cnt_da_rech90           0
```

```
fr_da_rech90          0
cnt_loans30           0
amnt_loans30          0
maxamnt_loans30       0
medianamnt_loans30    0
cnt_loans90           0
amnt_loans90          0
maxamnt_loans90       0
medianamnt_loans90    0
payback30             0
payback90             0
pcircle               0
pdate                 0
dtype: int64
```

```
train['pcircle'].value_counts()
```

```
UPW    209593
Name: pcircle, dtype: int64
```

```
train.drop('pcircle',axis=1,inplace=True) #Same value , so not much informative
```

```
# Checking for duplicate values
print("Number of duplicate values in train data is "+str(sum(train.duplicated())))
```

```
Number of duplicate values in train data is 1
```

Separating features and class labels

```
X = train
X = X.drop(["label"], axis = 1)
```

```
y = train['label']
```

```
X.shape , y.shape
```

```
((209593, 32), (209593,))
```

Checking Data Imbalances

```
#Pandas Series.value_counts() function return a Series containing counts of
#unique values. The resulting object will be in descending order so that the
#first element is the most frequently-occurring element. Excludes NA values by
#default

print(train['label'].value_counts())
f,ax=plt.subplots(1,2,figsize=(16,6))
labels = ['0', '1']
train['label'].value_counts().plot.pie(explode=[0,0.2],autopct='%1.1f%%',ax=ax[0],shadow=True,labels=labels,fontsize=1(
sns.countplot('label',data=train, ax=ax[1])
ax[1].set_xticklabels(['0', '1'], fontsize=10)
plt.show()
```

- Imbalanced Data

```
## SEE the number of of outliers

Q1 = train.quantile(0.25)
Q3 = train.quantile(0.75)
IQR = Q3 - Q1
print('No. of outliers in all the fields: ',((train < (Q1 - 1.5 * IQR)) | (train > (Q3 + 1.5 * IQR))).sum())
```

```
No. of outliers in all the fields:  amnt_loans30            10416
amnt_loans90            12590
aon                      3607
cnt_da_rech30            4114
cnt_da_rech90            5367
cnt_loans30              7817
cnt_loans90             11523
```

```
cnt_ma_rech30           11294
cnt_ma_rech90           14155
daily_decr30            16350
daily_decr90            18187
fr_da_rech30             1579
fr_da_rech90              865
fr_ma_rech30            11450
fr_ma_rech90            26845
label                   26162
last_rech_amt_ma        20864
last_rech_date_da        6732
last_rech_date_ma       20145
maxamnt_loans30         30400
maxamnt_loans90         28648
medianamnt_loans30      14148
medianamnt_loans90      12169
medianamnt_ma_rech30    24928
medianamnt_ma_rech90    25457
medianmarechprebal30    27252
medianmarechprebal90    25933
msisdn                      0
payback30               16532
payback90               17850
pdate                       0
rental30                18526
rental90                19399
sumamnt_ma_rech30       13219
sumamnt_ma_rech90       13954
dtype: int64
```

```
# Correlations
f, ax = plt.subplots(figsize=(20, 10))
sns.heatmap(train.corr(method='spearman'), annot=True, cmap="YlGnBu")
```

Convert all columns to numeric

```
for i in X.columns:
    if i=='pdate':
        continue
    else:
        X[i]=pd.to_numeric(X[i],errors='coerce')
```

```
train['msisdn'].value_counts()
```

```
04581I85330    7
47819I90840    7
29191I82738    6
43430I70786    6
71742I90843    6
              ..
06791I70785    1
09434I82730    1
65674I70370    1
76802I89231    1
18134I85330    1
Name: msisdn, Length: 186243, dtype: int64
```

```
X.drop(['msisdn','pdate'],axis=1,inplace=True) # Not much informative in this case
```

```
X = np.array(X)
```

Train Test Split

```
from sklearn.model_selection import train_test_split
#Split arrays or matrices into random train and test subsets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,test_size = 0.25,random_state = 42)
```

Standardize the features

```
#Use standardscaler to standardize the features

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_cv  = sc.transform(X_cv)
X_test  = sc.transform(X_test)
```

```
(len(X_train),len(y_train),len(X_test),len(y_test),len(X_cv),len(y_cv))
```

```
    (117895, 117895, 52399, 52399, 39299, 39299)
```

UTILITY FUNCTIONS

```
def plot_matrix(matrix,labels):#DEPRECATED: Function plot_confusion_matrix is
# deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
# ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
```

```
        plt.figure(figsize=(20,7))
        sns.heatmap(matrix, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
        plt.xlabel('Predicted Class')
        plt.ylabel('Original Class')
        plt.show()

    # This function plots the confusion matrices given y_i, y_i_hat.
    def plot_confusion_matrix(test_y, predict_y):
        cm = confusion_matrix(test_y, predict_y)
        # C = 9,9 matrix, each cell (i,j) represents number of points of class i
        #are predicted class j

        recall_table =(((cm.T)/(cm.sum(axis=1))).T)
        # How did we calculateed recall_table :
        # divide each element of the confusion matrix with the sum of elements in
        #that column
        # C = [[1, 2],
        #     [3, 4]]
        # C.T = [[1, 3],
        #        [2, 4]]
        # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to
        #rows in two diamensional array
        # C.sum(axix =1) = [[3, 7]]
        # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
        #                           [2/3, 4/7]]
        # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
        #                           [3/7, 4/7]]
        # sum of row elements = 1

        precision_table =(cm/cm.sum(axis=0))
        # How did we calculateed precision_table :
        # divide each element of the confusion matrix with the sum of elements in
        #that row
```

```
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to
    #rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [0,1]
    print()
    print("-"*20, "Confusion matrix", "-"*20)
    plot_matrix(cm,labels)

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plot_matrix(precision_table,labels)

    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plot_matrix(recall_table,labels)


#Data preparation for ML models.

#Misc. functionns for ML models

def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities
    #belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
```

```
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

```
Xr = np.array(X_test)
yr = np.array(y_test)
```

NOTE :

- Since we want a probabilistic interpretation from the model so we will use **LogLoss** as the Metric here

Prediction using a 'Random' Model

- We build a random model to compare the log- loss of random model with the ML models used by us.
- In a 'Random' Model, we generate the '2' class probabilites randomly such that they sum to 1.

```
# We need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their
#sum
# ref: https://stackoverflow.com/a/18662466/4084039
```

```
test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,2))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,2)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
# We create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,2))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,2)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

Logistic Regression with class balancing

```
alpha = [10 ** x for x in range(-6, 6)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ',
      alpha[best_alpha],
```

```
          "The train log loss is:",
          log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

    predict_y = sig_clf.predict_proba(X_cv)
    print('For values of best alpha = ',
          alpha[best_alpha],
          "The cross validation log loss is:",
          log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

    predict_y = sig_clf.predict_proba(X_test)
    print('For values of best alpha = ',
          alpha[best_alpha],
          "The test log loss is:",
          log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
#Linear classifiers (SVM, logistic regression, etc.) with SGD training.

#This estimator implements regularized linear models with stochastic gradient
# descent (SGD) learning: the gradient of the loss is estimated each sample at
#a time and the model is updated along the way with a decreasing strength
#schedule (aka learning rate). SGD allows minibatch (online/out-of-core)
#learning via the partial_fit method. For best results using the default
#learning rate schedule, the data should have zero mean and unit variance.
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

Test some points out

- Correctly predicted

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train,y_train)
test_point_index = 1
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1, -1)),4))
print("Actual Class :", yr[test_point_index].reshape(1, -1))
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3022 0.6978]]
Actual Class : [[1]]
```

- Incorrectly predicted

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train,y_train)
test_point_index = 5456
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1, -1)),4))
print("Actual Class :", yr[test_point_index].reshape(1, -1))
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3461 0.6539]]
Actual Class : [[0]]
```

Linear Support Vector Machines

```
alpha = [10 ** x for x in range(-6, 6)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))
```

```
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ',
      alpha[best_alpha],
      "The train log loss is:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ',
      alpha[best_alpha],
      "The cross validation log loss is:",
      log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ',
      alpha[best_alpha],
```

```
      "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
#Linear classifiers (SVM, logistic regression, etc.) with SGD training.

#This estimator implements regularized linear models with stochastic gradient
#descent (SGD) learning: the gradient of the loss is estimated each sample at a
#time and the model is updated along the way with a decreasing strength schedule
#(aka learning rate). SGD allows minibatch (online/out-of-core) learning via the
#partial_fit method. For best results using the default learning rate schedule,
# the data should have zero mean and unit variance.

predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

Test some points out

- Correctly Classified

```
# from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(X_train,y_train)
test_point_index = 1
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1, -1)),4))
print("Actual Class :", yr[test_point_index].reshape(1, -1))
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.0735 0.9265]]
Actual Class : [[1]]
```

- Incorrectly Classified

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(X_train,y_train)
test_point_index = 5456
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1, -1)),4))
print("Actual Class :", yr[test_point_index].reshape(1, -1))
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3741 0.6259]]
Actual Class : [[0]]
```

Random Forest

```
alpha = [100,300,500]
max_depth = [3, 5]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(X_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_train, y_train)
        sig_clf_probs = sig_clf.predict_proba(X_cv)
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(y_cv, sig_clf_probs))
```

```
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpl
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best estimator = ',
```

```
      alpha[int(best_alpha/2)],
      "The train log loss is:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(X_cv)
print('For values of best estimator = ',
      alpha[int(best_alpha/2)],
      "The cross validation log loss is:",
      log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(X_test)
print('For values of best estimator = ',
      alpha[int(best_alpha/2)],
      "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  3
Log Loss : 0.2784886966047876
for n_estimators = 100 and max depth =  5
Log Loss : 0.2669527019385133
for n_estimators = 300 and max depth =  3
Log Loss : 0.2782845046015356
for n_estimators = 300 and max depth =  5
Log Loss : 0.26715790109268056
for n_estimators = 500 and max depth =  3
Log Loss : 0.2785084940520751
for n_estimators = 500 and max depth =  5
Log Loss : 0.2672758826707215
For values of best estimator =  100 The train log loss is: 0.26520610571689845
For values of best estimator =  100 The cross validation log loss is: 0.2669527019385133
For values of best estimator =  100 The test log loss is: 0.27225935017934794
```

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpl
#A random forest classifier.
```

```
#A random forest is a meta estimator that fits a number of decision tree
#classifiers on various sub-samples of the dataset and uses averaging to improve
#the predictive accuracy and control over-fitting. The sub-sample size is
#controlled with the max_samples parameter if bootstrap=True (default),
#otherwise the whole dataset is used to build each tree.
predict_and_plot_confusion_matrix(X_train, y_train,X_cv,y_cv, clf)
```

Test some points out

- Correctly classified

```
test_point_index = 5
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1,-1)),4))
print("Actual Class :", yr[test_point_index].reshape(1,-1))
indices = np.argsort(-clf.feature_importances_)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.0428 0.9572]]
Actual Class : [[1]]
```

- Incorrectly Classified

```
test_point_index = 5456
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1,-1)),4))
print("Actual Class :", yr[test_point_index].reshape(1,-1))
indices = np.argsort(-clf.feature_importances_)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2557 0.7443]]
Actual Class : [[0]]
```

Let's try UPSAMPLING

```
# define oversampling strategy
from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy='minority')
# fit and apply the transform
X_over, y_over = oversample.fit_resample(X, y)
print('Before Upsampling',X.shape, ' ', y.shape)
print('After Upsampling',X_over.shape, ' ', y_over.shape)
```

```
Before Upsampling (209593, 32)   (209593,)
After Upsampling (366862, 32)   (366862,)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, test_size=0.25, random_state=42)
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,test_size = 0.25,random_state = 42)


#Use standardscaler to standardize the features

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_cv  = sc.transform(X_cv)
X_test  = sc.transform(X_test)
```

Logistic Regression

```
alpha = [10 ** x for x in range(-6, 6)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
```

```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ',
      alpha[best_alpha],
      "The train log loss is:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ',
      alpha[best_alpha],
      "The cross validation log loss is:",
      log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ',
      alpha[best_alpha],
      "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
#Linear classifiers (SVM, logistic regression, etc.) with SGD training.
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

- This was correctly classified before upsampling by all models

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train,y_train)
test_point_index = 1
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(Xr[test_point_index].reshape(1, -1)),4))
print("Actual Class :", yr[test_point_index].reshape(1, -1))
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
Predicted Class : 0
Predicted Class Probabilities: [[0.8069 0.1931]]
Actual Class : [[1]]
```

Lets summarize above models before proceeding with the feature engineering approach.

```
from prettytable import PrettyTable
#A table can be created with add_row or add_column methods. The example creates
# a PrettyTable with the add_row method. From the module, we import PrettyTable . We set the header names.

ptable = PrettyTable()
ptable.title = "*** Model Summary *** [Performance Metric: Log-Loss]"
ptable.field_names=["Model Name","Train LogLoss","CV LogLoss","Test LogLoss","% Misclassified Points"]
ptable.add_row(["Logistic Regression With Class balancing","0.298","0.297","0.302","0.122"])
ptable.add_row(["Linear SVM","0.309","0.308","0.312","0.122"])
ptable.add_row(["Random Forest Classifier ","0.265","0.266","0.272","0.095"])
ptable.add_row(["Logistic Regression With Class balancing(UPSAMPLING) ","0.522","0.521","0.522","0.247"])


print(ptable)
```

```
+----------------------------------------------------+---------------+------------+--------------+-----------
|                     Model Name                     | Train LogLoss | CV LogLoss | Test LogLoss | % Misclassi
+----------------------------------------------------+---------------+------------+--------------+-----------
|       Logistic Regression With Class balancing     |     0.298     |   0.297    |    0.302     |          0.1
|                     Linear SVM                     |     0.309     |   0.308    |    0.312     |          0.1
|              Random Forest Classifier              |     0.265     |   0.266    |    0.272     |          0.0
| Logistic Regression With Class balancing(UPSAMPLING) |     0.522     |   0.521    |    0.522     |          0.2
+----------------------------------------------------+---------------+------------+--------------+-----------
```

CONCLUSION:

- All the models performed better than the random model, which makes sense.
- From the pretty table we can see that , **RandomForest** performed best here.
- Even the overfitting is not present if we check the train and test logloss, they are very close
- Over sampling method was also applied on the training data to make the data more balanced, but it gave worse results

0s completed at 12:27