Delinquency Telecom Model

Definition:

- **Delinquency** is a condition that arises when an activity or situation does not occur at its scheduled (or expected) date i.e., it occurs later than expected.

Use Case:

- Many donors, experts, and microfinance institutions (MFI) have become convinced that using mobile financial services (MFS) is more convenient and efficient, and less costly, than the traditional high-touch model for delivering microfinance services. MFS becomes especially useful when targeting the unbanked poor living in remote areas. The implementation of MFS, though, has been uneven with both significant challenges and successes.
- Today, microfinance is widely accepted as a poverty-reduction tool, representing $70 billion in outstanding loans and a global outreach of 200 million clients.
- One of our Client in Telecom collaborates with an MFI to provide micro-credit on mobile balances to be paid back in 5 days. The Consumer is believed to be delinquent if he deviates from the path of paying back the loaned amount within 5 days

Machine Learning problem :

- 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')
- Basically a Binary Classification setup

Business objectives and constraints.

- No low-latency requirement for Paying Back loaned amount.
- Probability of a data-point belonging to each loan transaction is needed.

Performance Metric

- Log-loss (Since probabilities is our concern)
- Confusion matrix (Also want to check some precision and recalls)

```
!pip install catboost
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/47/80/8e9c57ec32dfed6ba2922bc5c964
     |████████████████████████████████| 67.3MB 40kB/s
```

```
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from catbo
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from cat
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
Installing collected packages: catboost
Successfully installed catboost-0.25.1
```

IMPORT NECESSARY LIBRARIES ALL REQUIRED LIBRARIES ARE AT FIRST LOADED PANDAS USED FOR THE DATA HANDLING MATPLOTLIB AND SEABORN USED FOR DATA VISUALIZATION Numpy Provides faster data handling and also important mathematical features

```
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_lo
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix, normalized_mutual_i
from sklearn.linear_model import SGDClassifier
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:
  warnings.warn(message, FutureWarning)
```

We should always target to increase sensitivity over specificity as sensitivity is the true positive rate whereas specificity is the false positive rate. *True Positive* : These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. *False Positive* : These are the wrongly predicted positive values which means that the value of actual class is no and the predicted class is yes. All the important parameters such as Accuracy, Precision and Recall depend upon our True positive rate.So if there is a increase in false positive rate then therewill be a downgrade in these parameters which in return will give not so good outcomes.

READ IN AND EXPLORE THE DATA

```
df = pd.read_csv('/content/sample_data_intw.csv')
```

```
df.head()
```

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26

```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

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

```
Size of data = (118071, 36)
```

Checks for Null values

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118071 entries, 0 to 118070
Data columns (total 36 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   label                 118071 non-null  int64  
 1   msisdn                118071 non-null  object 
 2   aon                   118071 non-null  float64
 3   daily_decr30          118071 non-null  float64
 4   daily_decr90          118071 non-null  float64
 5   rental30              118071 non-null  float64
 6   rental90              118071 non-null  float64
 7   last_rech_date_ma     118071 non-null  float64
 8   last_rech_date_da     118071 non-null  float64
 9   last_rech_amt_ma      118071 non-null  int64  
 10  cnt_ma_rech30         118071 non-null  int64  
 11  fr_ma_rech30          118071 non-null  float64
 12  sumamnt_ma_rech30     118071 non-null  float64
 13  medianamnt_ma_rech30  118071 non-null  float64
 14  medianmarechprebal30  118071 non-null  float64
 15  cnt_ma_rech90         118071 non-null  int64  
 16  fr_ma_rech90          118071 non-null  int64  
 17  sumamnt_ma_rech90     118071 non-null  int64  
 18  medianamnt_ma_rech90  118071 non-null  float64
 19  medianmarechprebal90  118071 non-null  float64
 20  cnt_da_rech30         118071 non-null  float64
 21  fr_da_rech30          118071 non-null  float64
 22  cnt_da_rech90         118071 non-null  int64  
 23  fr_da_rech90          118071 non-null  int64  
 24  cnt_loans30           118071 non-null  int64  
 25  amnt_loans30          118071 non-null  int64  
 26  maxamnt_loans30       118071 non-null  float64
 27  medianamnt_loans30    118071 non-null  float64
```

```
 28  cnt_loans90           118071 non-null  float64
 29  amnt_loans90          118071 non-null  int64
 30  maxamnt_loans90       118071 non-null  int64
 31  medianamnt_loans90    118071 non-null  float64
 32  payback30             118071 non-null  float64
 33  payback90             118071 non-null  float64
 34  pcircle               118070 non-null  object
 35  pdate                 118070 non-null  object
dtypes: float64(21), int64(12), object(3)
memory usage: 32.4+ MB
```

```
df.isnull().sum()
```

```
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                 1
pdate                   1
dtype: int64
```

```
print(df.dtypes)
#It Tells us about the data Types of all the feature in the data
```

```
label                     int64
msisdn                   object
aon                     float64
daily_decr30            float64
daily_decr90            float64
rental30                float64
rental90                float64
last_rech_date_ma       float64
last_rech_date_da       float64
last_rech_amt_ma          int64
cnt_ma_rech30             int64
fr_ma_rech30            float64
sumamnt_ma_rech30       float64
medianamnt_ma_rech30    float64
medianmarechprebal30    float64
cnt_ma_rech90             int64
fr_ma_rech90              int64
sumamnt_ma_rech90         int64
medianamnt_ma_rech90    float64
medianmarechprebal90    float64
cnt_da_rech30           float64
fr_da_rech30            float64
cnt_da_rech90             int64
fr_da_rech90              int64
cnt_loans30               int64
amnt_loans30              int64
maxamnt_loans30         float64
medianamnt_loans30      float64
cnt_loans90             float64
amnt_loans90              int64
maxamnt_loans90           int64
medianamnt_loans90      float64
payback30               float64
payback90               float64
pcircle                  object
pdate                    object
dtype: object
```

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

```
# Checking for duplicate values
print("Number of duplicate values in data set is "+str(sum(df.dup
```

```
Number of duplicate values in data set is 0
```

Separating features and class labels

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

```
y = df['label']
```

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

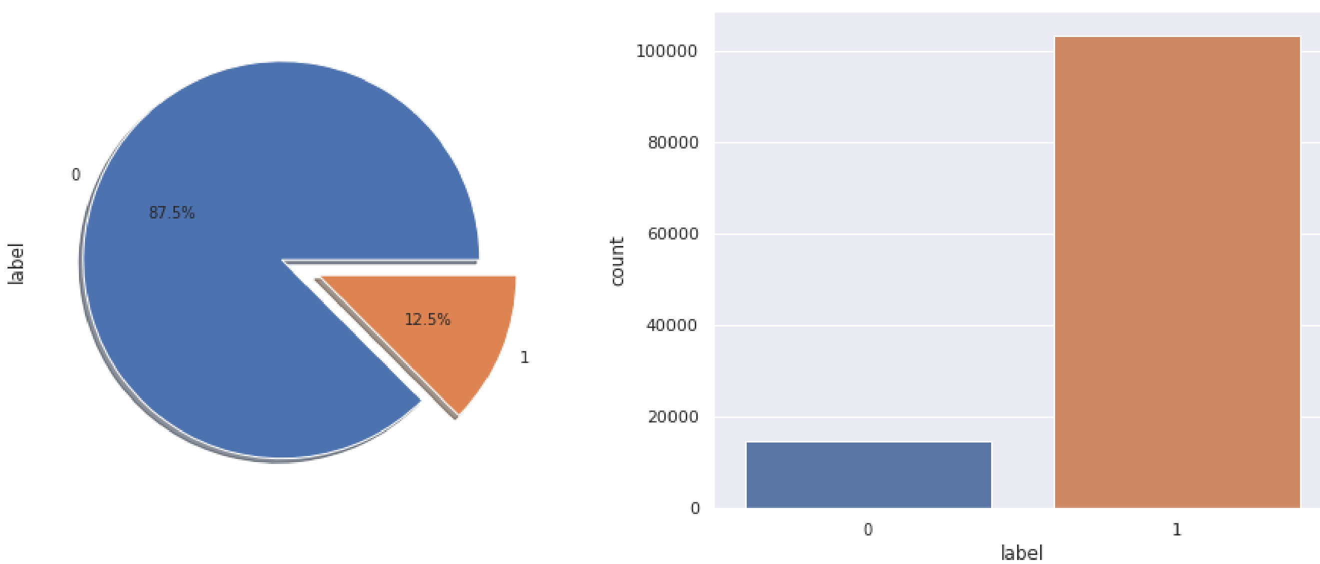
```
((118071, 34), (118071,))
```

DATA Visulazation

Checking Data Imbalances

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

```
1    103326
0     14745
Name: label, dtype: int64
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass tl
  FutureWarning
```



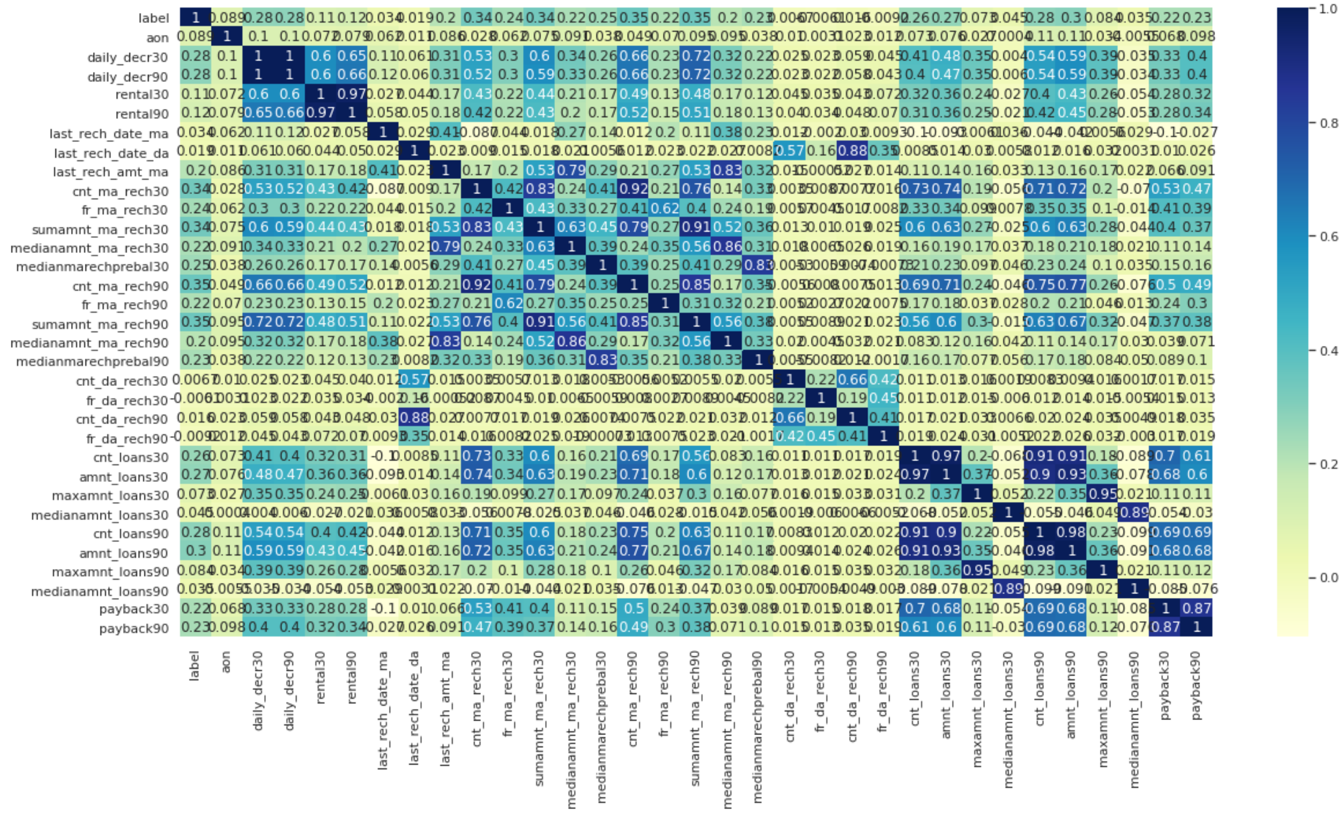
- Imbalanced Data

```
## See the number of of outliers

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print('No. of outliers in all the fields: ',((df < (Q1 - 1.5 * IQ
```

```
No. of outliers in all the fields:  amnt_loans30            5835
amnt_loans90             7102
aon                      2048
cnt_da_rech30            2344
cnt_da_rech90            3057
cnt_loans30              4373
cnt_loans90              6482
cnt_ma_rech30            6271
cnt_ma_rech90            7960
daily_decr30             9236
daily_decr90            10283
fr_da_rech30              922
fr_da_rech90              515
fr_ma_rech30             6399
fr_ma_rech90            15116
label                   14745
last_rech_amt_ma        11794
last_rech_date_da        3854
last_rech_date_ma       11335
maxamnt_loans30         17019
maxamnt_loans90         16015
medianamnt_loans30       8041
medianamnt_loans90       6954
medianamnt_ma_rech30    14117
medianamnt_ma_rech90    14402
medianmarechprebal30    15256
medianmarechprebal90    14615
msisdn                      0
payback30                9304
payback90               10023
pdate                       0
rental30                10539
rental90                11028
sumamnt_ma_rech30        7374
sumamnt_ma_rech90        7814
dtype: int64
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f53eb5d6990>
```



DATA CLEANING

Convert all columns to numeric

```
for i in X.columns:
    if i=='pdate':
        continue
    else:
```

```
        X[i]=pd.to_numeric(X[i],errors='coerce')
```

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

```
42825I88688    6
30080I90588    6
78160I89231    5
45099I84456    5
78109I96341    5
              ..
23225I90588    1
04397I85328    1
58944I89239    1
10120I88648    1
09988I70374    1
Name: msisdn, Length: 110088, dtype: int64
```

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

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

Train Test Split

```
#We used the normalizer to stop the spread of the data and Normal
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,test_
```

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(
```

```
(66414, 66414, 29518, 29518, 22139, 22139)
```

UTILITY FUNCTIONS

```
def plot_matrix(matrix,labels):
    plt.figure(figsize=(20,7))
    sns.heatmap(matrix, annot=True, cmap="YlGnBu", fmt=".3f", xti
    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

    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
    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #         [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 co
    # 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
    # C = [[1, 2],
    #      [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 co
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                       [3/4, 4/6]]
```

```
    #                           [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, te
    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 prob
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(tes
    # calculating the number of data points that are misclassifie
    print("Number of mis-classified points :", np.count_nonzero((
    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)
```

Double-click (or enter) to edit

 # Feature Selection

NOTE :

- Since we want a prediction probabilistic interpretation from the model under one of the two classes(1 or 0). so we will use **LogLoss** as the Metric here
- **Prediction Probability:*** The binary classification algorithms First predict probability for a recored to be classified under class (1 or 0) based on whether the probability crossed a threshold value,which is usually set at 0.5 by default.

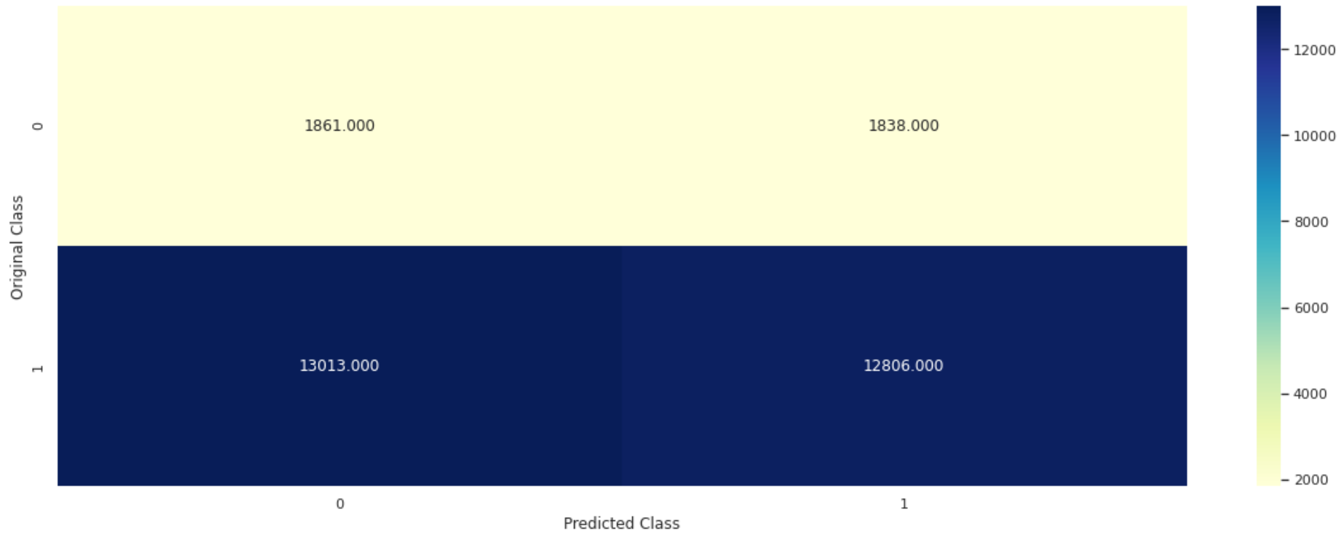
Prediction using a 'Random' Model

- 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
# one solution is to genarate 9 numbers and divide each of the nu
# 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 d
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_


# 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])
```

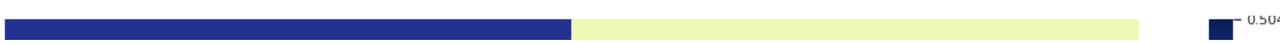
```
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,

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

```
Log loss on Cross Validation Data using Random Model 0.8871633966784495
Log loss on Test Data using Random Model 0.8878708721364589

-------------------- Confusion matrix --------------------
```

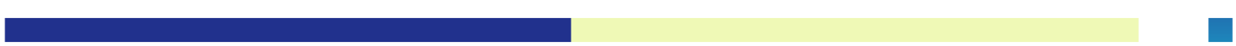



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

We found that the class of 0 was highly Imbalanced the minority class was about 12.50% and thats why most of the prediction model was predicting higher values and to handel this situation we used the concept of Up sampling the data by putting duplicate values of Minority Class



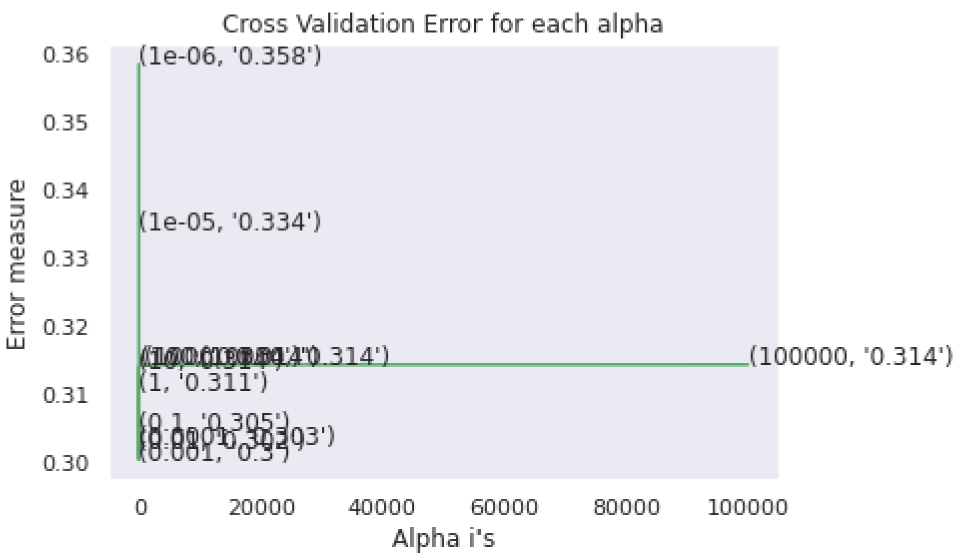
Logistic Regression with class balancing



```
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
    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, label
    # to avoid rounding error while multiplying probabilites we u
    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
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_alp
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)
```

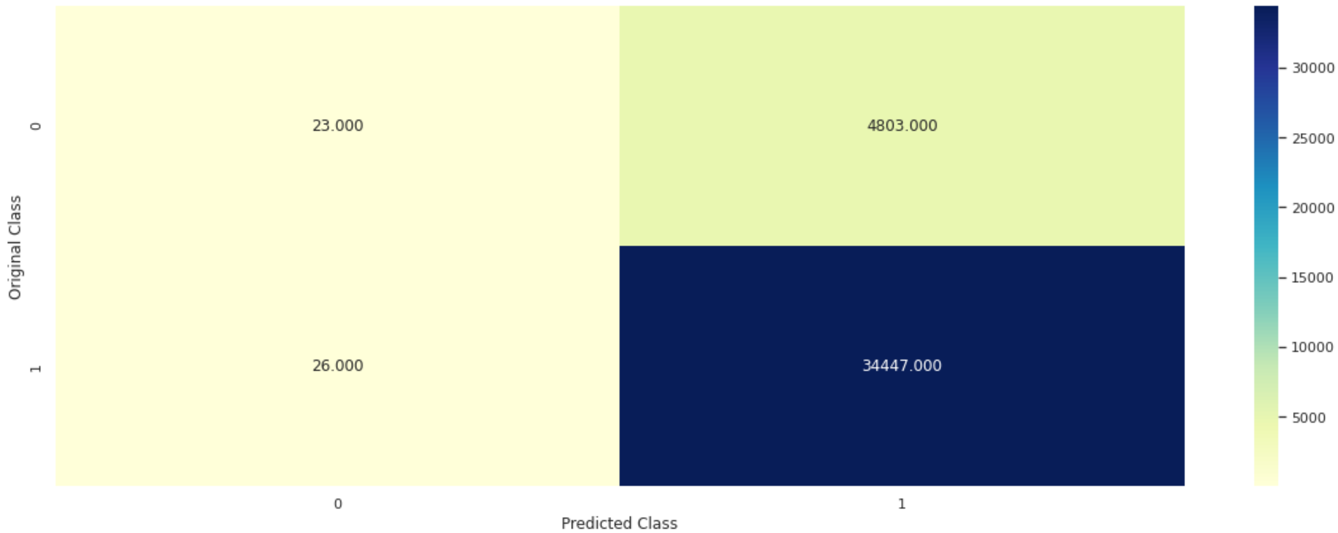
```
for alpha = 1e-06
Log Loss : 0.3583741973056819
for alpha = 1e-05
Log Loss : 0.33410751428057683
for alpha = 0.0001
Log Loss : 0.3025418160550398
for alpha = 0.001
Log Loss : 0.3003022735560267
for alpha = 0.01
Log Loss : 0.30184894206857465
for alpha = 0.1
Log Loss : 0.3047759284929228
for alpha = 1
Log Loss : 0.3105108900439599
for alpha = 10
Log Loss : 0.31364846764333576
for alpha = 100
Log Loss : 0.31420571193782953
for alpha = 1000
Log Loss : 0.31423102336865655
for alpha = 10000
Log Loss : 0.3142707025037973
for alpha = 100000
Log Loss : 0.31429166983926654
```



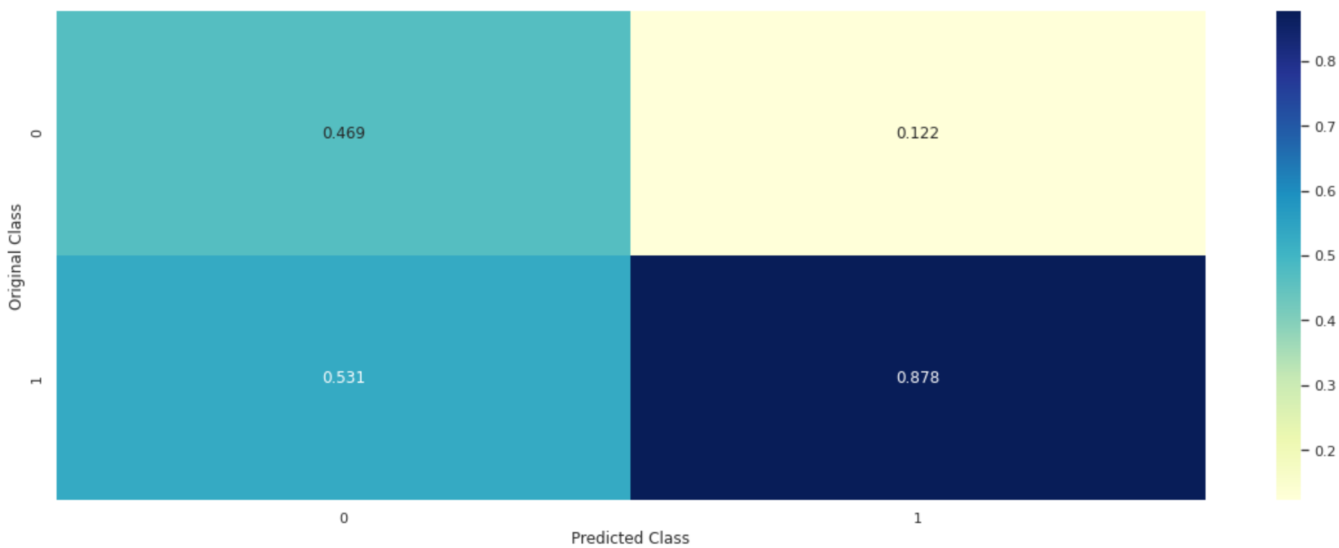

```
For values of best alpha =  0.001 The train log loss is: 0.29612631948598334
For values of best alpha =  0.001 The cross validation log loss is: 0.3003022735560267
For values of best alpha =  0.001 The test log loss is: 0.29743187269258287
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, c
```

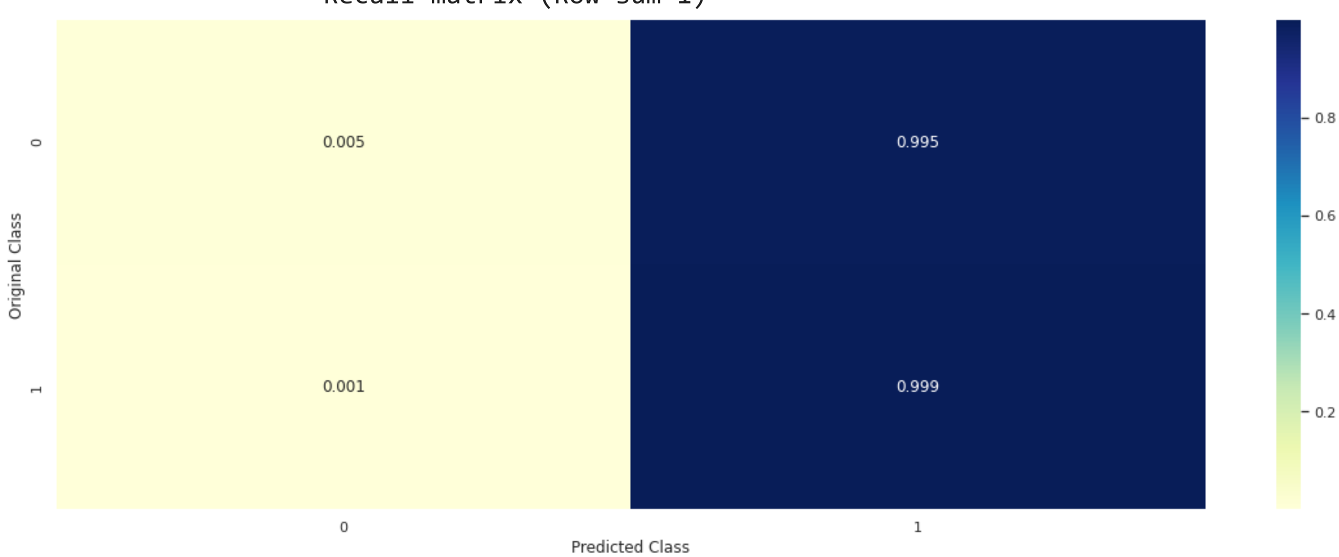
```
Log loss : 0.2979462032962917
Number of mis-classified points : 0.12287844474414107

-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



TRAINING THE MODEL

Test some points out

- Correctly predicted

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
clf.fit(X_train,y_train)
test_point_index = 1
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_
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: [[0.0912 0.9088]]
Actual Class : [[1]]
```

- Incorrectly predicted

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
clf.fit(X_train,y_train)
```

```
test_point_index = 5456
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_
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: [[0.1113 0.8887]]
Actual Class : [[1]]
```

Linear Support Vector Machines

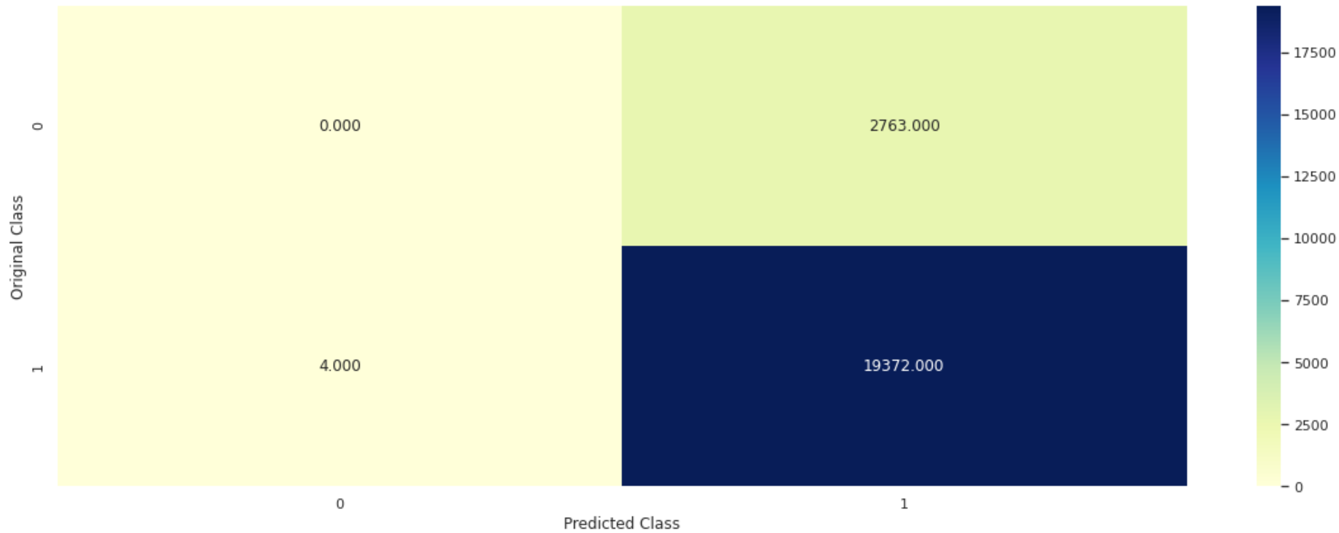
```
alpha = [10 ** x for x in range(-6, 6)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', rand
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, label
    # to avoid rounding error while multiplying probabilites we u
    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
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='
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)
```

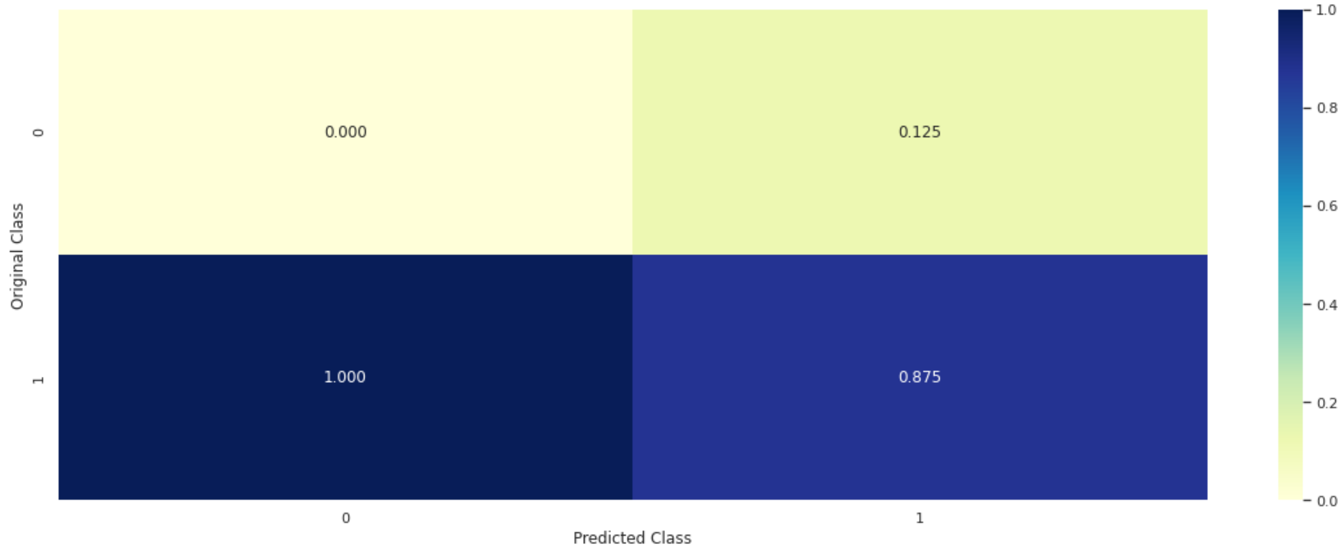
```
for alpha = 1e-06
Log Loss : 0.34395057375737187
for alpha = 1e-05
Log Loss : 0.3390516018714481
for alpha = 0.0001
Log Loss : 0.3363538213923128
for alpha = 0.001
Log Loss : 0.33324361292062726
for alpha = 0.01
Log Loss : 0.3303836736464843
for alpha = 0.1
Log Loss : 0.33260342421136674
for alpha = 1
Log Loss : 0.32363033193450313
for alpha = 10
Log Loss : 0.32933096538432255
for alpha = 100
Log Loss : 0.31428186191941815
for alpha = 1000
Log Loss : 0.3142809598463374
```

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, c
```

```
Log loss : 0.3142809598463374
Number of mis-classified points : 0.12498306156556303

-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

Test some points out

- Correctly Classified

```
# from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='
clf.fit(X_train,y_train)
test_point_index = 1
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_
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: [[0.1059 0.8941]]
Actual Class : [[1]]
```

- Incorrectly Classified

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
clf.fit(X_train,y_train)
test_point_index = 5456
no_feature = 1000
predicted_cls = sig_clf.predict(Xr[test_point_index].reshape(1, -
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_
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: [[0.1786 0.8214]]
Actual Class : [[1]]
```

Random Forest

```
alpha = [100,300,500]
max_depth = [3, 5]
cv_log_error_array = []
for i in alpha:
```

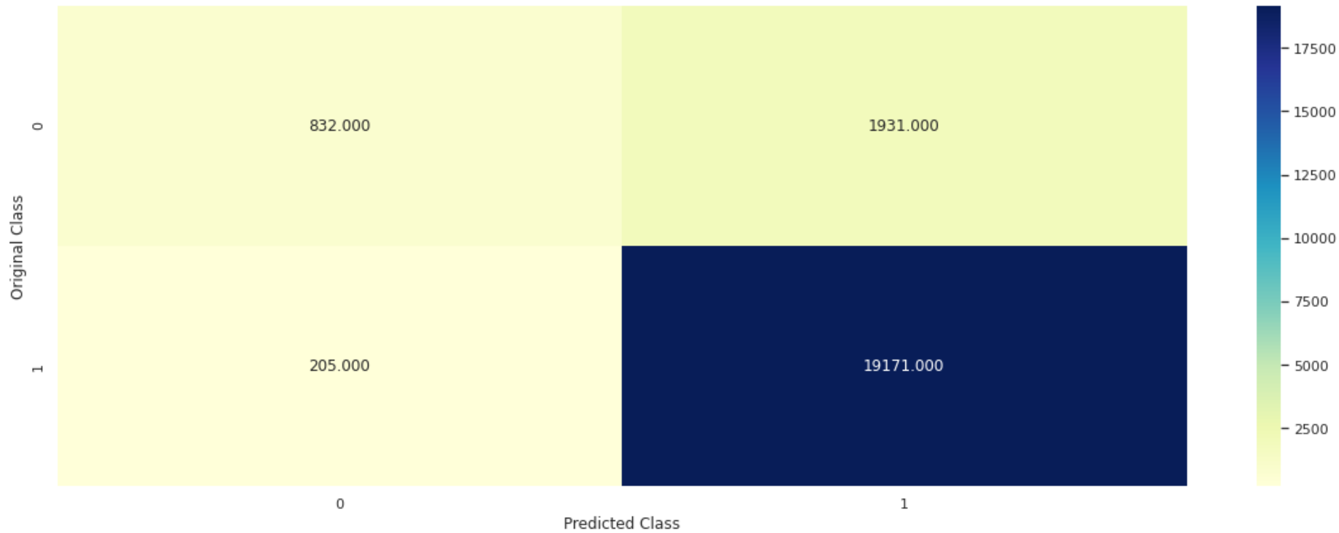
```
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='g
        clf.fit(X_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_train, y_train)
        sig_clf_probs = sig_clf.predict_proba(X_cv)
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, l
        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)
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 : 0.28354675337331575
for n_estimators = 100 and max depth =  5
Log Loss : 0.2732493763465254
for n_estimators = 300 and max depth =  3
```

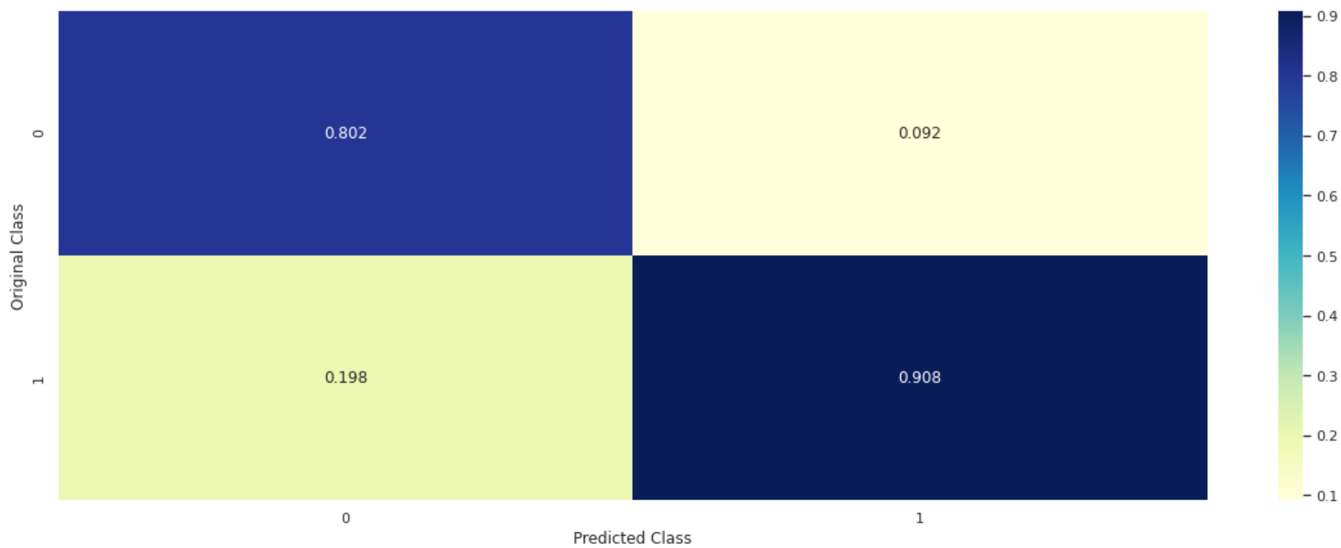
```
Log Loss : 0.28319479127447267
for n_estimators = 300 and max depth =  5
Log Loss : 0.27323811601108217
for n_estimators = 500 and max depth =  3
Log Loss : 0.283182145780339
for n_estimators = 500 and max depth =  5
Log Loss : 0.27317627166287023
For values of best estimator =  500 The train log loss is: 0.2637736113269699
For values of best estimator =  500 The cross validation log loss is: 0.27317627166287023
For values of best estimator =  500 The test log loss is: 0.2698448220751733
```

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)
predict_and_plot_confusion_matrix(X_train, y_train,X_cv,y_cv, clf
```

```
Log loss : 0.27317627168796
Number of mis-classified points : 0.09648132255296084

-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

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_
```

```
print("Actual Class :", yr[test_point_index].reshape(1,-1))
indices = np.argsort(-clf.feature_importances_)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.0602 0.9398]]
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_
print("Actual Class :", yr[test_point_index].reshape(1,-1))
indices = np.argsort(-clf.feature_importances_)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.1698 0.8302]]
Actual Class : [[1]]
```

Let's try UP SAMPLING

```
# 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 (118071, 32)   (118071,)
After Upsampling (206652, 32)   (206652,)
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning: The r
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_over, y_ove
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,test

#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

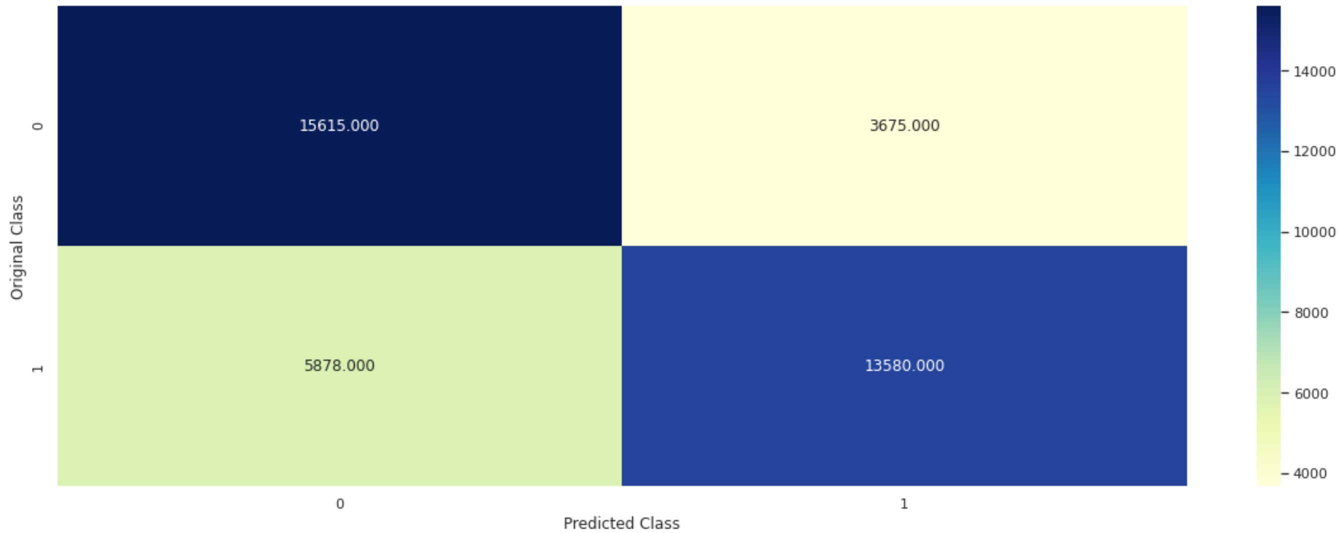
```
alpha = [10 ** x for x in range(-6, 6)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, label
    # to avoid rounding error while multiplying probabilites we u
    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
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_alp
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
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)
```

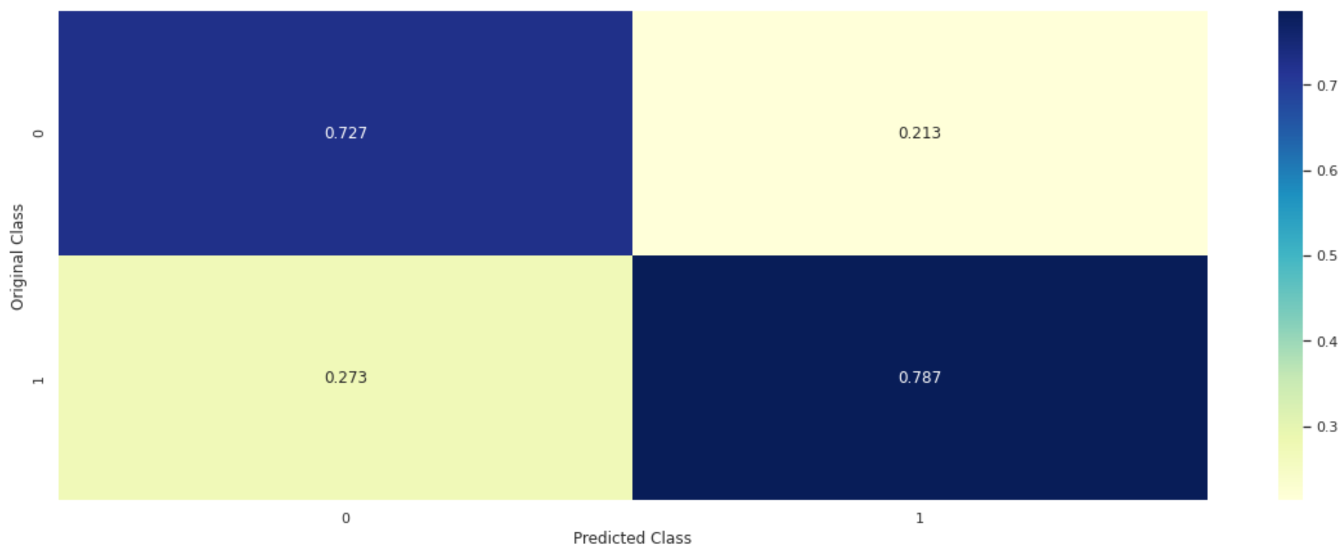
```
for alpha = 1e-06
Log Loss : 0.612615615994828
for alpha = 1e-05
Log Loss : 0.567703874160378
for alpha = 0.0001
Log Loss : 0.5198259991424637
for alpha = 0.001
Log Loss : 0.5168390361389236
for alpha = 0.01
Log Loss : 0.5219835040382559
for alpha = 0.1
Log Loss : 0.5298448273540146
for alpha = 1
Log Loss : 0.5421367708663628
for alpha = 10
Log Loss : 0.5489877786109534
for alpha = 100
Log Loss : 0.5501495983756718
for alpha = 1000
Log Loss : 0.5502747272531426
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, c
```

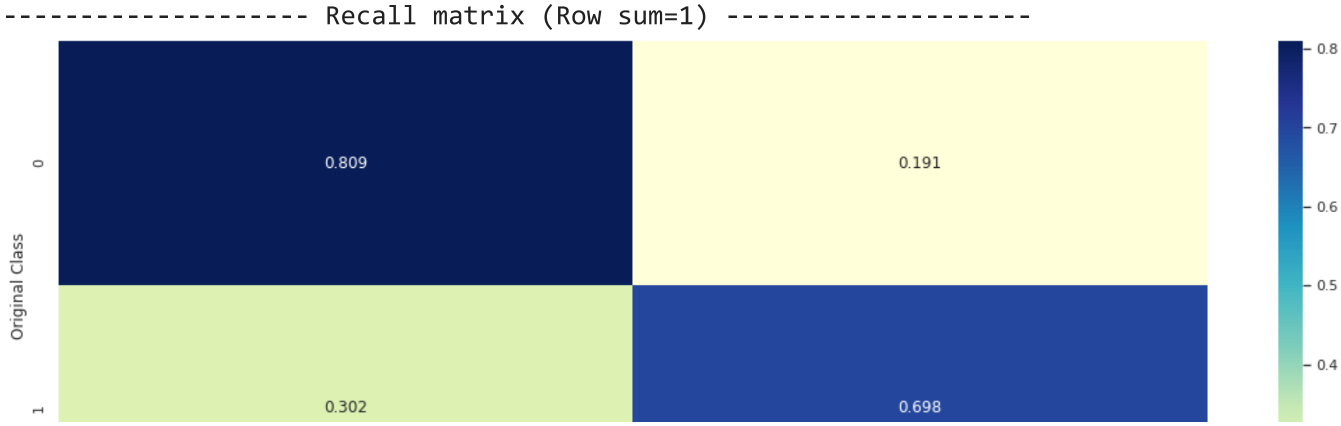
```
Log loss : 0.5168390361389236
Number of mis-classified points : 0.24654175699390937

-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



- This was correctly classified before upsampling by all models


```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alp
clf.fit(X_train,y_train)
```