Business Problem

Problem Description

We all know that when we visit an e-commerce or TV series website or even YouTube we see a separate suggestion box, where in they show some content which you might like. These are mainly based on the content that you have consumed on their website previously. These are called as Recommendation engine.

Now consider you have been running a start up since last one year and now you have been able to gather some customer data and you want to build a recommendation engine. Based on certain features you have to cluster the customers into two different groups so that you can recommend the correct products based on the customer's cluster.

Problem Statement

to build a predictive model to predict the category of the customer based on certain set of features

Real world/Business Objectives and constraints

Objectives:

1. Predict the Catefory of Cutomer.
2. increase the Precision (macro precision)

Constraints:

1. Some form of interpretability.

Machine Learning Problem

Data Overview

Column Description

1)customer_visit_score: a score based on how regularly the customer visits the website.

2)customer_product_search_score: quality or price range of product that the customer searches for.For ex: a customer searching for a laptop will have more weightage than someone looking for a book.

3)customer_ctr_score: how many of the searched links does the customer click.

4)customer_stay_score: a score based on the time spent on an avg. by the customer.

5)customer_frequency_score: a score based on how many times in a day the customer visit the website.

6)customer_product_variation_score: a score based on how many varities of products does a customer search for,for ex. electronics, apparels, etc.

7)customer_order_score: Score based on the no. of orders that has been succesfully delivered and not returned.

8)customer_affinity_score: an internal overall score calculated which signifies the affinity of the customer towards the website.

9)customer_category: the cluster/group to which the customer should belong to

10)customer_active_segment: the categorization of the customers based on their activity

11)X_1: Anonymized feature based on loyalty of the customer

Mapping the real world problem to a Machine Learning Problem

Type of Machine Learning Problem

```
1)For a given implicit data of user we need to predict the category/cluster that user
belongs to.
2)The given problem is a classification based Recommendation problem
```

Performance metric

```
1) Macro Precision (given)
2) confusion matrix (for better understanding)
```

Machine Learning Objective and Constraints

```
1)maximize the precision
2)provide some form of interpretability
```

In [1]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import os
import pandas as pd
import math as m
from sklearn.preprocessing import OneHotEncoder
```

In [2]:

```
df= pd.read_csv("Train.csv")
```

In [3]:

```
df["customer_category"].value_counts()
```

Out[3]:

```
0    9443
1    1295
Name: customer_category, dtype: int64
```

In [4]:

```
df
```

Out[4]:

	customer_id	customer_visit_score	customer_product_search_score	customer_ctr_score	customer_stay_score	customer_frequer
0	csid_1	13.168425	9.447662	-0.070203	-0.139541	
1	csid_2	17.092979	7.329056	0.153298	-0.102726	
2	csid_3	17.505334	5.143676	0.106709	0.262834	
3	csid_4	31.423381	4.917740	-0.020226	-0.100526	

	customer_id	customer_visit_score	customer_product_search_score	customer_ctr_score	customer_stay_score	customer_frequer
4	csid_5	11.909502	4.237073	0.187178	0.172891	
...	...	...	...	...	...	
10733	csid_10734	23.672615	6.701514	0.092879	-0.017332	
10734	csid_10735	25.673028	6.497796	0.050216	-0.047211	
10735	csid_10736	31.676844	7.799880	0.062961	-0.032765	
10736	csid_10737	28.441780	5.588302	-0.093931	0.081586	
10737	csid_10738	20.663035	4.478301	0.253165	0.381349	

10738 rows × 12 columns

In [5]:

```
df.dtypes
```

Out[5]:

```
customer_id                          object
customer_visit_score                float64
customer_product_search_score       float64
customer_ctr_score                  float64
customer_stay_score                 float64
customer_frequency_score            float64
customer_product_variation_score    float64
customer_order_score                float64
customer_affinity_score             float64
customer_active_segment              object
X1                                   object
customer_category                     int64
dtype: object
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10738 entries, 0 to 10737
Data columns (total 12 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   customer_id                       10738 non-null  object
 1   customer_visit_score              10738 non-null  float64
 2   customer_product_search_score     10696 non-null  float64
 3   customer_ctr_score                10738 non-null  float64
 4   customer_stay_score               10701 non-null  float64
 5   customer_frequency_score          10738 non-null  float64
 6   customer_product_variation_score  10692 non-null  float64
 7   customer_order_score              10672 non-null  float64
 8   customer_affinity_score           10738 non-null  float64
 9   customer_active_segment           10715 non-null  object
 10  X1                                10701 non-null  object
 11  customer_category                 10738 non-null  int64
dtypes: float64(8), int64(1), object(3)
memory usage: 1006.8+ KB
```

In [7]:

```
df.describe()
```

Out[7]:

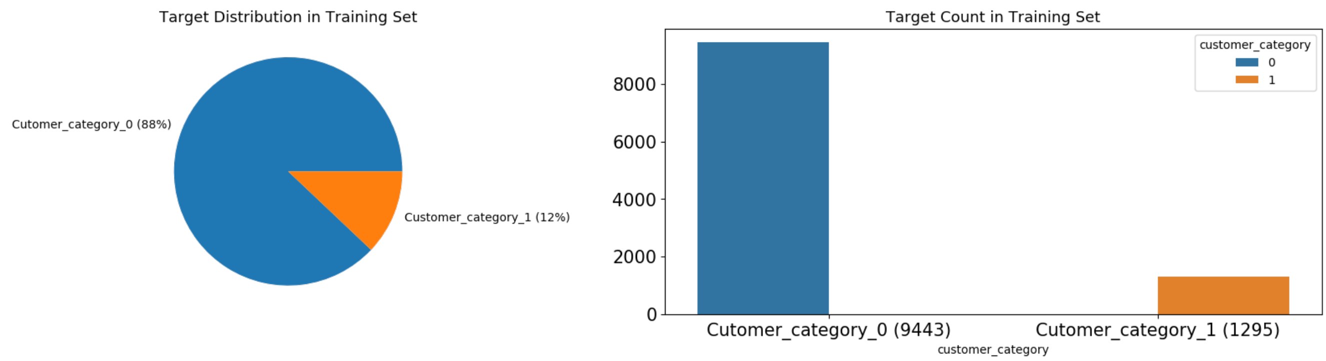
	customer_visit_score	customer_product_search_score	customer_ctr_score	customer_stay_score	customer_frequency_score	cu
count	10738.000000	10696.000000	10738.000000	10701.000000	10738.000000	
mean	19.060941	5.274847	0.175912	0.374230	2.376895	
std	7.419609	1.882559	0.372829	1.222031	5.601911	
min	0.568965	-0.161940	-0.547989	-0.462494	0.028575	

	customer_visit_score	customer_product_search_score	customer_ctr_score	customer_stay_score	customer_frequency_score	cu
25%	13.518021	3.971587	0.010840	0.027666	0.313610	
50%	18.774109	5.218479	0.074078	0.037201	0.516830	
75%	24.501719	6.520364	0.159606	0.179029	1.125380	
max	47.306691	16.638243	2.679474	14.701914	52.395014	

Exploratory Data Analysis

In [6]:

```
fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)
plt.tight_layout()

df.groupby('customer_category').count()['customer_id'].plot(kind='pie', ax=axes[0], labels=['Cutome
r_category_0 (88%)', 'Customer_category_1 (12%)'])
sns.countplot(x=df['customer_category'], hue=df['customer_category'], ax=axes[1])

axes[0].set_ylabel('')
axes[1].set_ylabel('')
axes[1].set_xticklabels(['Cutomer_category_0 (9443)', 'Cutomer_category_1 (1295)'])
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Target Distribution in Training Set', fontsize=13)
axes[1].set_title('Target Count in Training Set', fontsize=13)

plt.show()
```



observtions

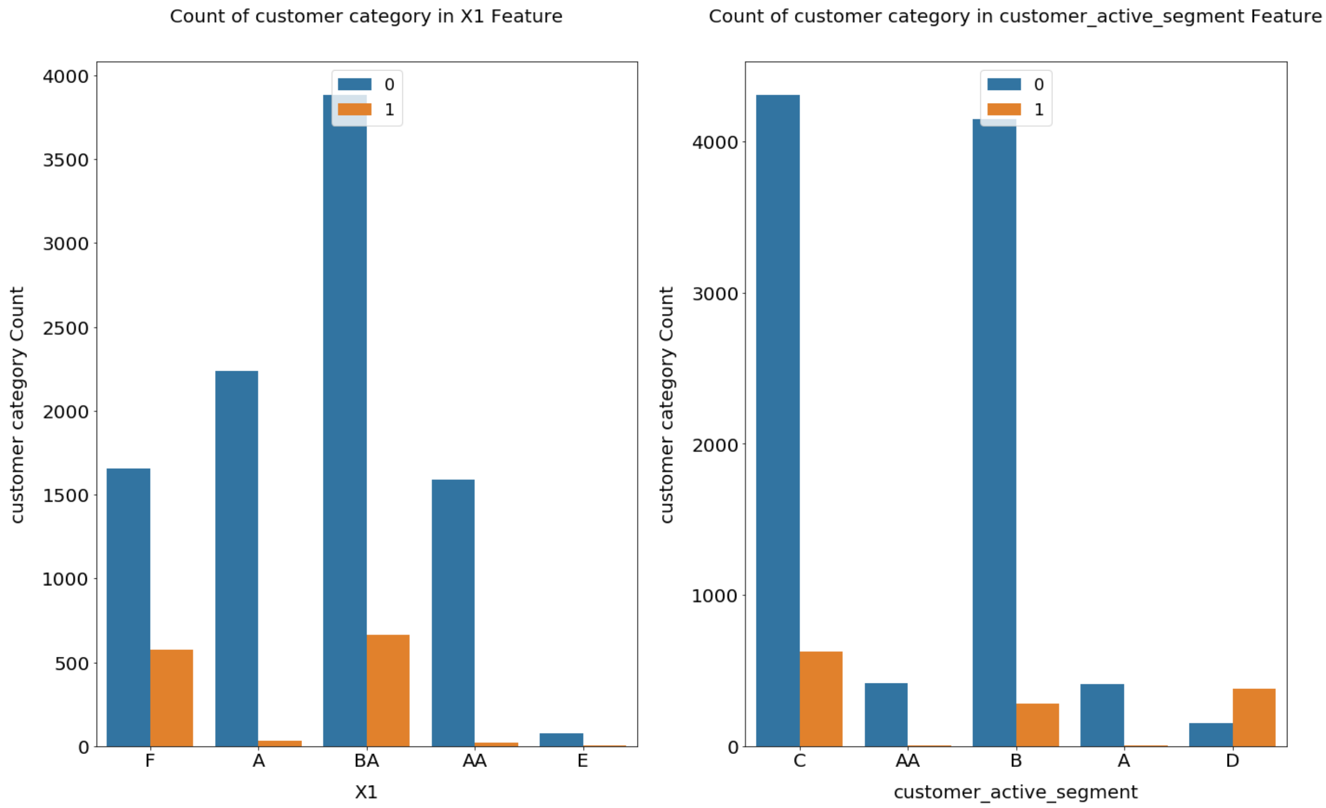
1)imabalanced data

Distributions of categorical variables

In [26]:

```
cat_features = ['X1','customer_active_segment']

fig, axs = plt.subplots(ncols=2, nrows=1, figsize=(20, 20))
plt.subplots_adjust(right=1.5, top=1.25)

for i, feature in enumerate(cat_features, 1):
    plt.subplot(2, 3, i)
    sns.countplot(x=feature, hue='customer_category', data=df)

    plt.xlabel('{}'.format(feature), size=20, labelpad=15)
    plt.ylabel('customer category Count', size=20, labelpad=15)
    plt.tick_params(axis='x', labelsize=20)
    plt.tick_params(axis='y', labelsize=20)

    plt.legend(['0', '1'], loc='upper center', prop={'size': 18})
    plt.title('Count of customer category in {} Feature'.format(feature), size=20, y=1.05)

plt.show()
```

```
plt.show()
```



Filling Missing/Null values

In [4]:

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

Out[4]:

```
customer_id                        0
customer_visit_score               0
customer_product_search_score     42
customer_ctr_score                 0
customer_stay_score               37
customer_frequency_score           0
customer_product_variation_score  46
customer_order_score              66
customer_affinity_score            0
customer_active_segment           23
X1                                37
customer_category                  0
dtype: int64
```

Handling X1

handling type of X1 based on customer_active_segment and category

In [8]:

```
df.groupby(["customer_category","customer_active_segment"])["X1"].value_counts()
```

Out[8]:

```
customer_category  customer_active_segment  X1
0                  A                        BA     164
                                            F       91
                                            A       89
                                            AA      64
                                            E        1
```

```
                    AA                        BA      179
                                              A        86
                                              F        80
                                              AA       63
                                              E         6
                    B                         BA     1713
                                              A       991
                                              AA      734
                                              F       659
                                              E        32
                    C                         BA     1719
                                              A      1032
                                              F       799
                                              AA      698
                                              E        34
                    D                         BA       71
                                              A        32
                                              F        26
                                              AA       25
                                              E         1
1                   A                         A         1
                                              BA        1
                    AA                        BA        1
                                              F         1
                    B                         BA      151
                                              F       106
                                              A        13
                                              AA       12
                                              E         1
                    C                         BA      313
                                              F       284
                                              A        16
                                              AA        9
                                              E         1
                    D                         BA      191
                                              F       185
                                              A         2
                                              AA        1
Name: X1, dtype: int64
```

for all pairs of categories and customer active segment "BA" occured the most

In [9]:

```
df["X1"].fillna('BA', inplace=True)
```

handling customer_active_segment

handling type of customer_active_segment based on X1 and category

In [10]:

```
df.groupby(["customer_category","X1"])["customer_active_segment"].value_counts()
```

Out[10]:

```
customer_category  X1   customer_active_segment
0                  A    C                          1032
                        B                           991
                        A                            89
                        AA                           86
                        D                            32
                   AA   B                           734
                        C                           698
                        A                            64
                        AA                           63
                        D                            25
                   BA   C                          1732
                        B                          1730
                        AA                          181
                        A                           165
```

```
                    D                           71
               E    C                           34
                    B                           32
                    AA                           6
                    A                            1
                    D                            1
               F    C                          799
                    B                          659
                    A                           91
                    AA                          80
                    D                           26
1              A    C                           16
                    B                           13
                    D                            2
                    A                            1
               AA   B                           12
                    C                            9
                    D                            1
               BA   C                          314
                    D                          193
                    B                          152
                    A                            1
                    AA                           1
               E    B                            1
                    C                            1
               F    C                          284
                    D                          185
                    B                          106
                    AA                           1
Name: customer_active_segment, dtype: int64
```

for all pairs of categories and X1 "C" occured the most except when x1 is "AA" where B occured most

In [11]:

```
df.loc[(df["X1"]=="AA")&(df["customer_active_segment"].isnull()==True),"customer_active_segment"]=
"B"
df["customer_active_segment"].fillna("C", inplace=True)
```

handling customer_product_variation_score

taking median of customer_product_variation_score based on combination of "customer_category","X1","customer_active_segment"

In [12]:

```
temp=df.groupby(["customer_category","X1","customer_active_segment"])
["customer_product_variation_score"].median()
```

In [13]:

```
temp[(1, 'A', 'AA')]=0.0
temp[(1, 'AA', 'A')]=0.0
```

In [14]:

```
for i in df.loc[df["customer_product_variation_score"].isnull()==True].index:
    #cc=df.loc[i,"customer_category"]
    x1=df.loc[i,"X1"]
    cas=df.loc[i,"customer_active_segment"]
    mean=(temp.loc[(0,x1,cas)]+temp.loc[(1,x1,cas)])/2
    df.loc[i,"customer_product_variation_score"]=mean
```

handling customer_order_score

taking median of customer_order_score based on combination of "customer_category","X1","customer_active_segment"

In [15]:

```
temp=df.groupby(["customer_category","X1","customer_active_segment"])["customer_order_score"].medi
an()
```

In [16]:

```
temp[(1, 'A', 'AA')]=0.0
temp[(1, 'AA', 'A')]=0.0
```

In [17]:

```
for i in df.loc[df["customer_order_score"].isnull()==True].index:
    #cc=df.loc[i,"customer_category"]
    x1=df.loc[i,"X1"]
    cas=df.loc[i,"customer_active_segment"]
    mean=(temp.loc[(0,x1,cas)]+temp.loc[(1,x1,cas)])/2
    df.loc[i,"customer_order_score"]=mean
```

handling customer_stay_score

taking median of customer_stay_score based on combination of "customer_category","X1","customer_active_segment"

In [18]:

```
temp=df.groupby(["customer_category","X1","customer_active_segment"])["customer_stay_score"].media
n()
```

In [19]:

```
temp
```

Out[19]:

```
customer_category  X1  customer_active_segment
0                  A   A                         -0.001653
                       AA                        -0.013515
                       B                          0.027801
                       C                          0.030292
                       D                          0.120880
                   AA  A                         -0.008224
                       AA                        -0.013584
                       B                          0.033496
                       C                          0.039688
                       D                          0.228429
                   BA  A                         -0.016936
                       AA                        -0.016099
                       B                          0.014965
                       C                          0.022216
                       D                          0.225245
                   E   A                         -0.007597
                       AA                         0.013317
                       B                          0.042916
                       C                          0.036424
                       D                          0.605800
                   F   A                         -0.011741
                       AA                        -0.017312
                       B                          0.005750
                       C                          0.010567
                       D                          0.262918
1                  A   A                          0.017433
                       B                          0.244170
                       C                          0.182308
                       D                          0.415579
                   AA  B                          0.366470
                       C                          0.137504
```

```
                    D                                   2.161326
                BA  A                                  -0.040917
                    AA                                  0.059243
                    B                                   0.741127
                    C                                   1.695612
                    D                                   3.688371
                E   B                                   0.019172
                    C                                   0.121890
                F   AA                                  0.473793
                    B                                   0.678976
                    C                                   2.150116
                    D                                   3.646187
Name: customer_stay_score, dtype: float64
```

In [20]:

```
temp[(1, 'A', 'AA')]=0.0
temp[(1, 'AA', 'A')]=0.0
temp[(1, 'AA', 'AA')]=0.0
```

In [21]:

```
for i in df.loc[df["customer_stay_score"].isnull()==True].index:
    #cc=df.loc[i,"customer_category"]
    x1=df.loc[i,"X1"]
    cas=df.loc[i,"customer_active_segment"]
    mean=(temp.loc[(0,x1,cas)]+temp.loc[(1,x1,cas)])/2
    df.loc[i,"customer_stay_score"]=mean
```

handling customer_product_search_score

taking median of customer_product_search_score based on combination of "customer_category","X1","customer_active_segment"

In [22]:

```
temp=df.groupby(["customer_category","X1","customer_active_segment"])
["customer_product_search_score"].median()
```

In [23]:

```
temp[(1, 'A', 'AA')]=0.0
temp[(1, 'AA', 'A')]=0.0
temp[(1, 'AA', 'AA')]=0.0
```

In [24]:

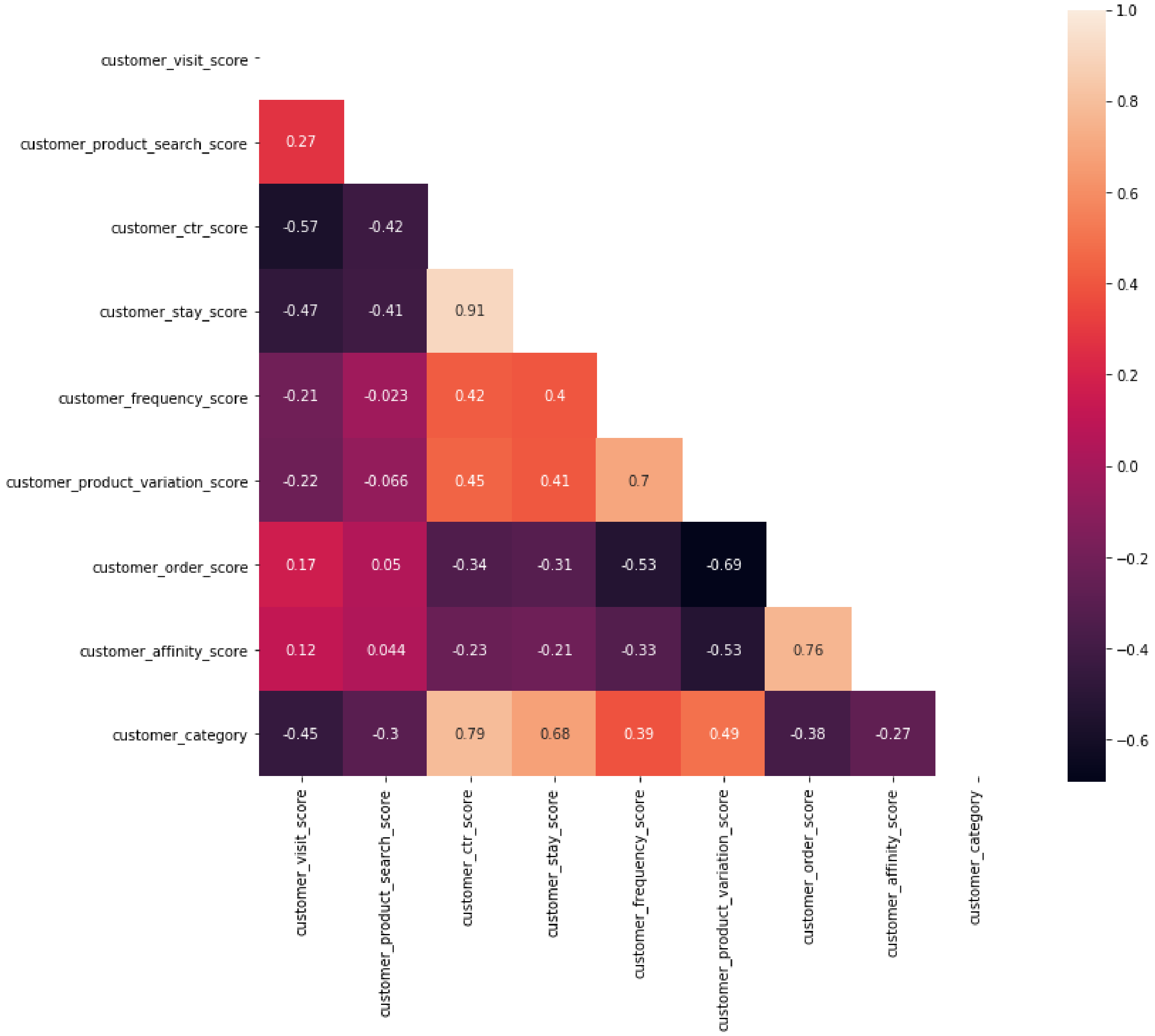
```
for i in df.loc[df["customer_product_search_score"].isnull()==True].index:
    #cc=df.loc[i,"customer_category"]
    x1=df.loc[i,"X1"]
    cas=df.loc[i,"customer_active_segment"]
    mean=(temp.loc[(0,x1,cas)]+temp.loc[(1,x1,cas)])/2
    df.loc[i,"customer_product_search_score"]=mean
```

Correlation matrix

In [25]:

```
corr = df.corr()
# Set up a mask
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(12, 10))
# Generate a custom diverging colormap
#cmap = sns.diverging_palette(230, 20, as_cmap=True)
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, annot=True, square=True)
```

```
plt.show()
```



observation

customer ctr score and customer stay score are highly correlated hence once of them can be removed

One hot encoded categorical variables

In [27]:

```
from sklearn.preprocessing import LabelEncoder

le1 = LabelEncoder()
le2 = LabelEncoder()

df['X1']= le1.fit_transform(df['X1'])
df['customer_active_segment']= le2.fit_transform(df['customer_active_segment'])
```

In [28]:

```
cat_features = ['X1', 'customer_active_segment']
encoded_features = []


for feature in cat_features:
        encoded_feat = OneHotEncoder().fit_transform(df[feature].values.reshape(-1, 1)).toarray()
        n = df[feature].nunique()
```

```
        cols = ['{}_{}'.format(feature, n) for n in range(1, n + 1)]
        encoded_df = pd.DataFrame(encoded_feat, columns=cols)
        encoded_df.index = df.index
        encoded_features.append(encoded_df)
df= pd.concat([df, *encoded_features[:2]], axis=1)
```

In [29]:

```
df.drop(['customer_id','customer_active_segment','X1'],axis=1,inplace=True)
```

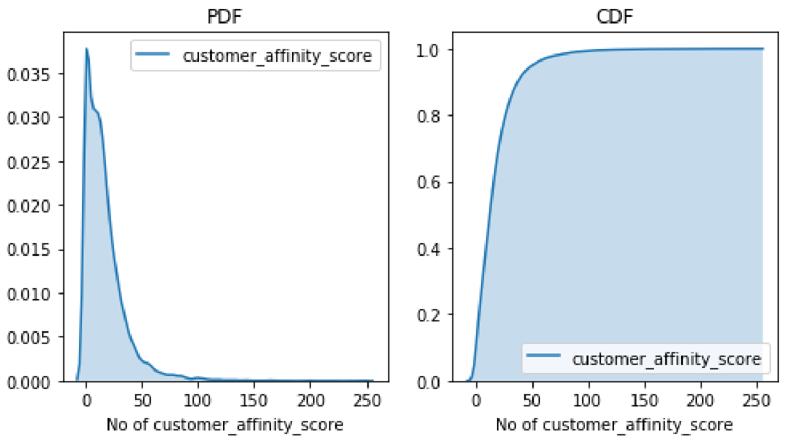
Univariate analysis

Boxplots, CDF and pdf

customer_affinity_score

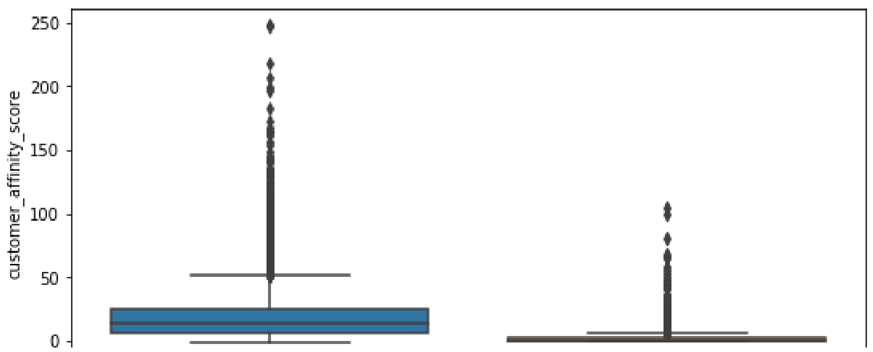
In [30]:

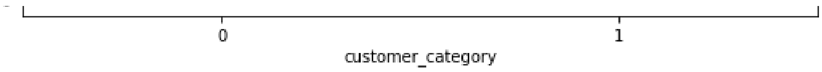
```
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)
sns.kdeplot(df.customer_affinity_score, shade=True, ax=ax1)
plt.xlabel('No of customer_affinity_score')
plt.title("PDF")

ax2 = plt.subplot(122)
sns.kdeplot(df.customer_affinity_score, shade=True, cumulative=True,ax=ax2)
plt.xlabel('No of customer_affinity_score')
plt.title('CDF')

plt.show()
```



In [31]:

```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_affinity_score', x='customer_category', data=df)
plt.show()
```





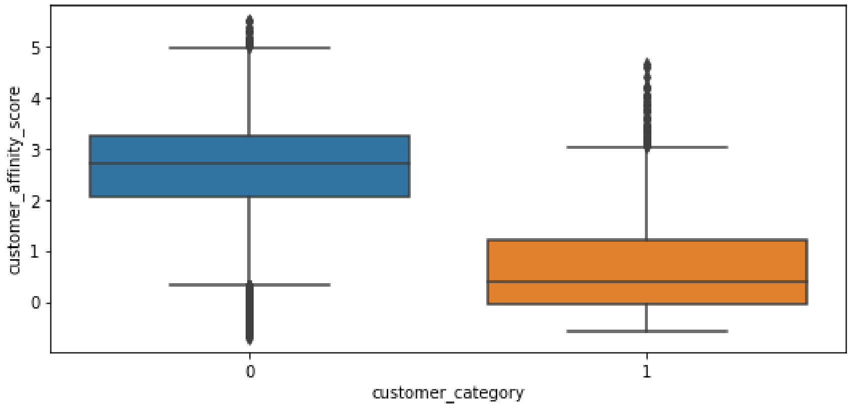
log transformation of customer_affinity_score

In [32]:

```
df["customer_affinity_score"]=np.log(df["customer_affinity_score"]+1)
```

In [33]:

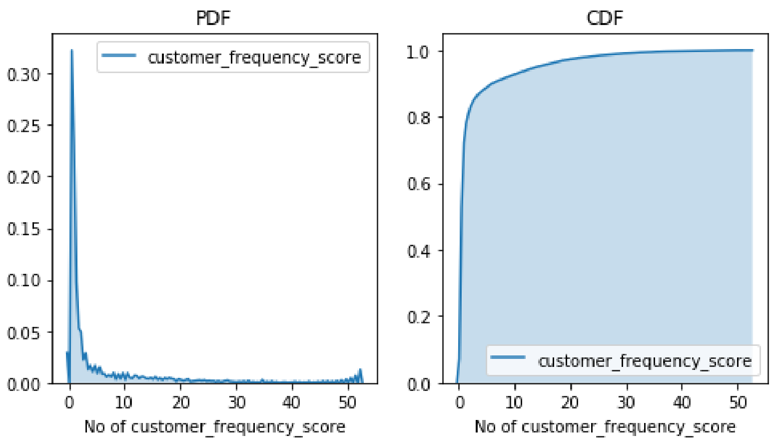
```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_affinity_score', x='customer_category', data=df)
plt.show()
```



customer_frequency_score

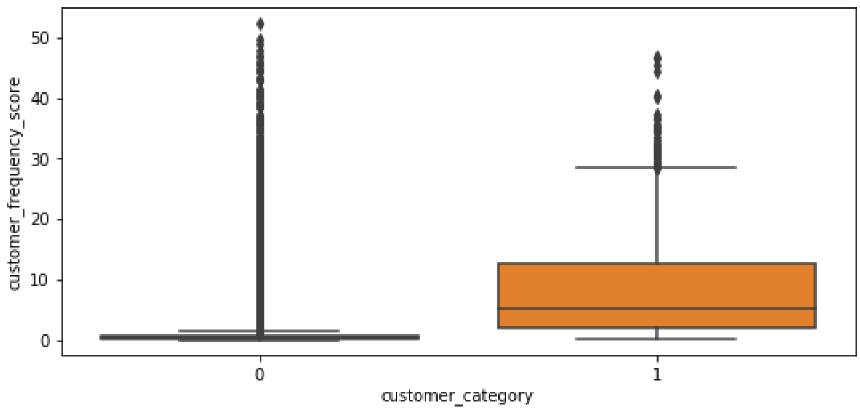
In [34]:

```
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)
sns.kdeplot(df.customer_frequency_score, shade=True, ax=ax1)
plt.xlabel('No of customer_frequency_score')
plt.title("PDF")

ax2 = plt.subplot(122)
sns.kdeplot(df.customer_frequency_score, shade=True, cumulative=True,ax=ax2)
plt.xlabel('No of customer_frequency_score')
plt.title('CDF')

plt.show()
```



In [35]:

```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_frequency_score', x='customer_category', data=df)
plt.show()
```



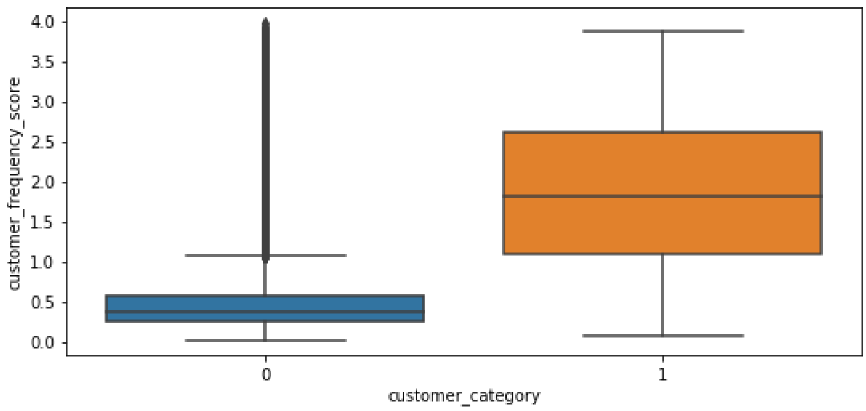
log transformation of customer_affinity_score

In [36]:

```
df["customer_frequency_score"]=np.log(df["customer_frequency_score"]+1)
```

In [37]:

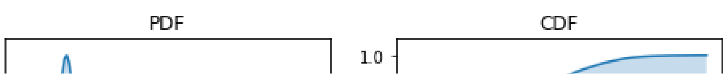
```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_frequency_score', x='customer_category', data=df)
plt.show()
```

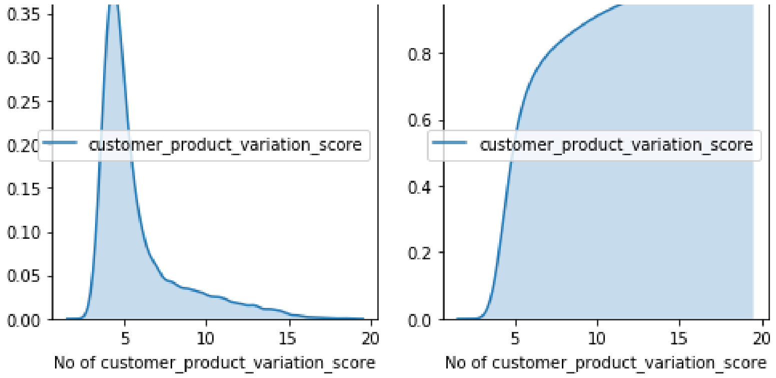


customer_product_variation_score

In [38]:

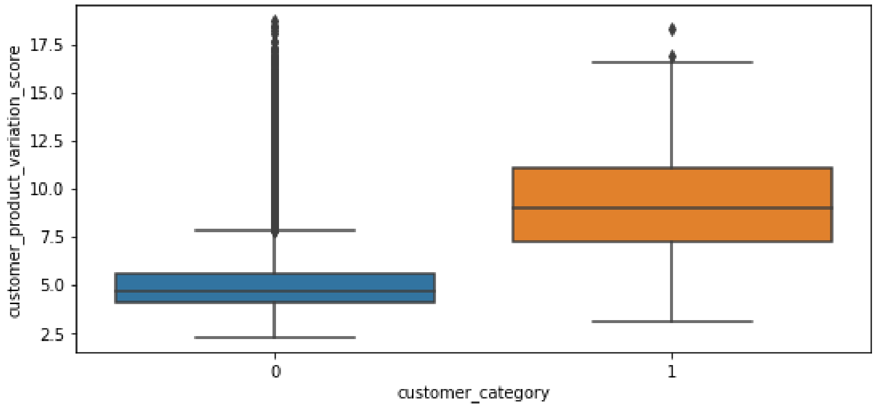
```
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)
sns.kdeplot(df.customer_product_variation_score, shade=True, ax=ax1)
plt.xlabel('No of customer_product_variation_score')
plt.title("PDF")

ax2 = plt.subplot(122)
sns.kdeplot(df.customer_product_variation_score, shade=True, cumulative=True,ax=ax2)
plt.xlabel('No of customer_product_variation_score')
plt.title('CDF')

plt.show()
```





In [39]:

```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_product_variation_score', x='customer_category', data=df)
plt.show()
```



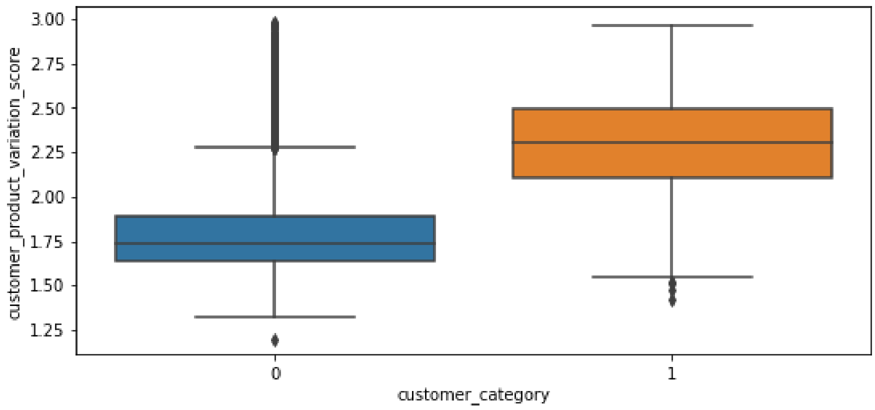
log transformation of customer_product_variation_score

In [40]:

```
df["customer_product_variation_score"]=np.log(df["customer_product_variation_score"]+1)
```

In [41]:

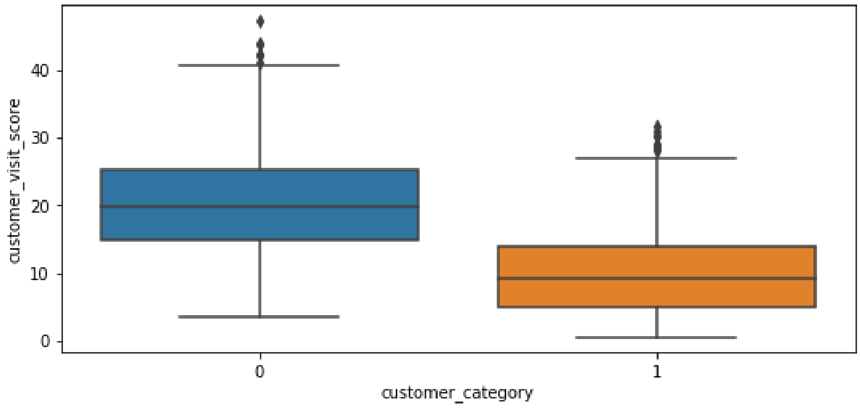
```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_product_variation_score', x='customer_category', data=df)
plt.show()
```



customer_visit_score

In [42]:

```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_visit_score', x='customer_category', data=df)
plt.show()
```



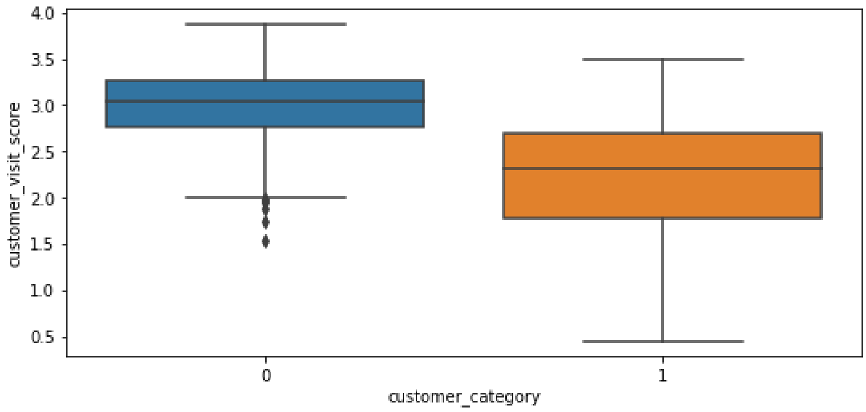
log transformation of customer_visit_score

In [43]:

```
df["customer_visit_score"]=np.log(df["customer_visit_score"]+1)
```

In [44]:

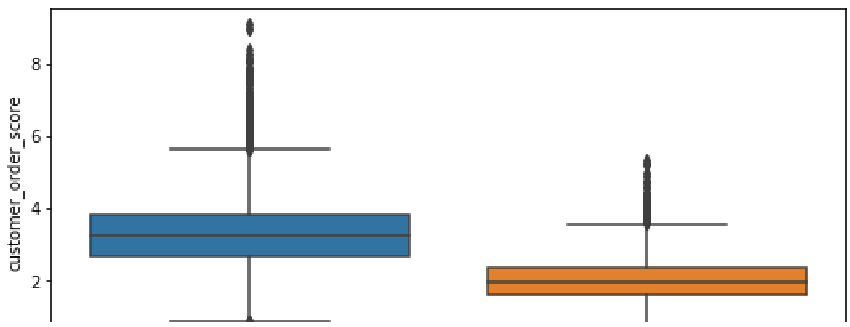
```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_visit_score', x='customer_category', data=df)
plt.show()
```

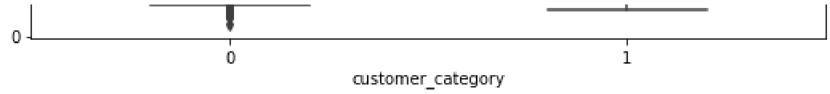


customer_order_score

In [45]:

```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_order_score', x='customer_category', data=df)
plt.show()
```





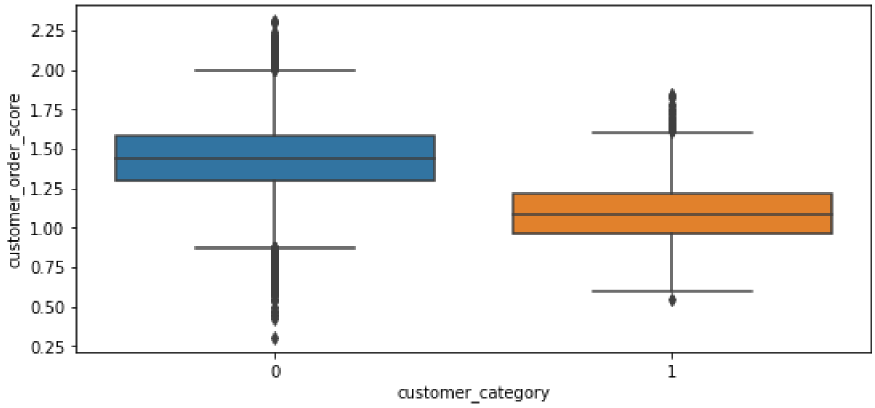
log transformation of customer_order_score

In [46]:

```
df["customer_order_score"]=np.log(df["customer_order_score"]+1)
```

In [47]:

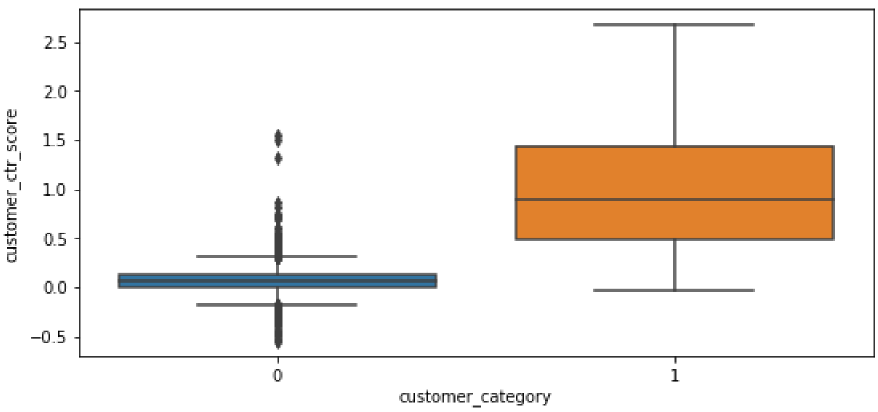
```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_order_score', x='customer_category', data=df)
plt.show()
```



customer_ctr_score

In [48]:

```
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='customer_ctr_score', x='customer_category', data=df)
plt.show()
```



Bi Variate analysis

In [26]:

```
sns.pairplot(df, hue="customer_category")
```

Out[26]:

```
<seaborn.axisgrid.PairGrid at 0x7feaa02794f0>
```

In [33]:

```
df.drop(["customer_visit_score","customer_product_search_score","customer_stay_score","X1_2","X1_3
","X1_5","customer_active_segment_1","customer_active_segment_3","customer_active_segment_5"],axis
=1,inplace=True)
```

Model

In [50]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
from collections import Counter
```

In [51]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
```

```
    C = 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

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

    B =(C/C.sum(axis=0))
    #divid 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]]
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [52]:

```
y=df["customer_category"]
df.drop(["customer_category"],axis=1,inplace=True)
x=df[:]
```

data splitting

In [53]:

```
X_train,X_test, y_train, y_test = train_test_split(x, y, stratify=y, test_size=0.3)
```

In [54]:

```
y_test.unique()
```

Out[54]:

```
array([0, 1])
```

In [55]:

```
X_train
```

Out[55]:

	customer_visit_score	customer_product_search_score	customer_ctr_score	customer_stay_score	customer_frequency_score	cu
4643	2.507528	8.874623	0.116326	-0.065969	0.185887	
5162	3.314022	6.668653	0.091264	0.020566	0.652554	
2599	3.020243	4.948821	-0.028001	-0.034535	0.241831	
5412	3.287233	6.580033	0.047034	-0.142731	0.223712	
3076	2.959934	2.232572	0.227571	0.371020	0.524403	
...	...	...	...	...	...	
6292	3.061734	2.904589	0.190220	0.149582	0.383522	
7497	3.343086	6.067079	-0.002276	-0.014912	0.447966	
10340	2.836932	7.803463	0.453654	0.355037	2.609363	
8887	3.077232	9.311759	0.382151	0.221405	2.973731	
9822	2.666056	2.145305	0.049689	0.236467	0.582290	

7516 rows × 18 columns

In [56]:

```
print("Number of data points in train data :",X_train.shape)
print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (7516, 18)
Number of data points in test data : (3222, 18)
```

In [57]:

```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len)
print("-"*10, "Distribution of output variable in test data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.8794571580627993 Class 1:  0.12054284193720063
---------- Distribution of output variable in test data ----------
Class 0:  0.1207324643078833 Class 1:  0.1207324643078833
```

logistic regression

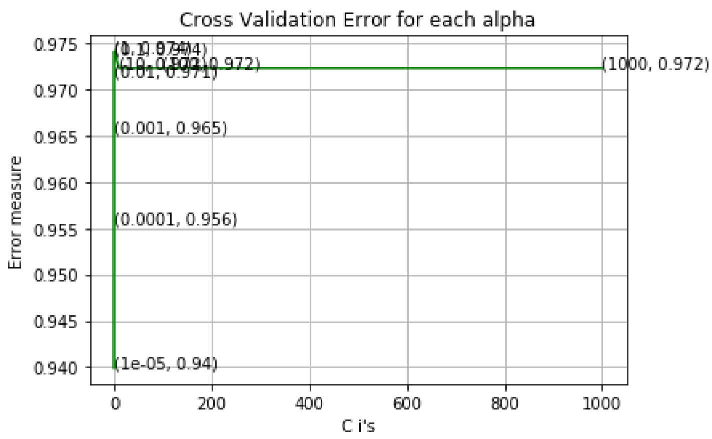
since data is highly imbalance to improve macro precision used classweight parameter to provide weightage to class

In [58]:

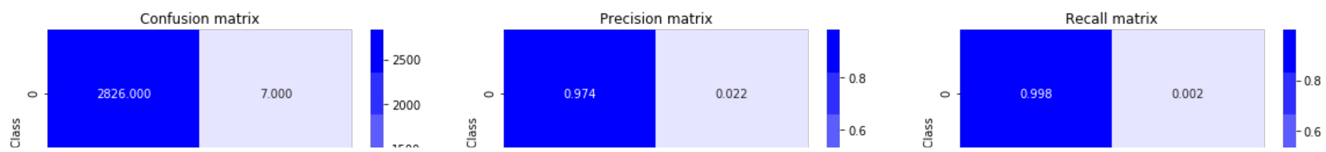
```
alpha = [10 ** x for x in range(-5, 4)]



log_error_array=[]
for i in alpha:
    clf = LogisticRegression(C=i,max_iter=10000,class_weight={1:12,0:88})#to get get better macro p
```

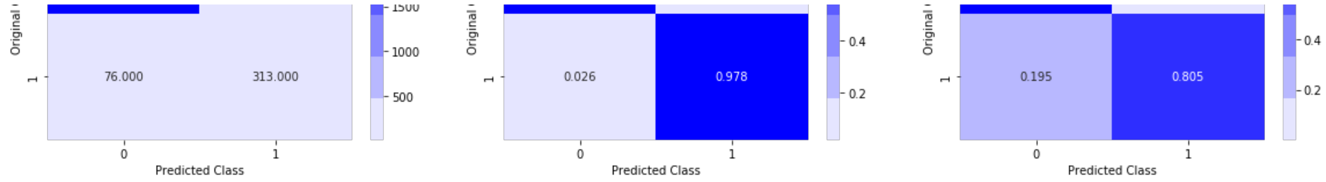
```
recision
    clf.fit(X_train, y_train)
    predict_y = clf.predict(X_test)
    log_error_array.append(precision_score(y_test, predict_y, average='macro'))
    print('For values of C = ', i, "The precision_score is:",precision_score(y_test, predict_y, ave
rage='macro'))

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


best_alpha = np.argmax(log_error_array)
clf = LogisticRegression(C=alpha[best_alpha],max_iter=10000)
clf.fit(X_train, y_train)
#sig_clf = CalibratedClassifierCV(clf, method="sigmoid")


predict_y = clf.predict(X_train)
print(set(y_train) - set(predict_y))
print('For values of best C = ', alpha[best_alpha], "The train precision_score
is:",precision_score(y_train, predict_y, average='macro'))
predict_y = clf.predict(X_test)
print(set(y_test) - set(predict_y))
print('For values of best C = ', alpha[best_alpha], "The test precision_score is:",precision_score
(y_test, predict_y, average='macro'))
#predicted_y =np.argmax(predict_y,axis=0)
print("Total number of data points :", len(predict_y))
plot_confusion_matrix(y_test, predict_y)
```

```
For values of C =  1e-05 The precision_score is: 0.9399068322981367
For values of C =  0.0001 The precision_score is: 0.9555207804657981
For values of C =  0.001 The precision_score is: 0.9653702616472618
For values of C =  0.01 The precision_score is: 0.9713970371487527
For values of C =  0.1 The precision_score is: 0.9738642745730157
For values of C =  1 The precision_score is: 0.9741463110667996
For values of C =  10 The precision_score is: 0.9723281127146844
For values of C =  100 The precision_score is: 0.9723281127146844
For values of C =  1000 The precision_score is: 0.9723281127146844
```




```
set()
For values of best C =  1 The train precision_score is: 0.9642930344792014
set()
For values of best C =  1 The test precision_score is: 0.9759680823569952
Total number of data points : 3222
```





In [59]:

```
clf.coef_
```

Out[59]:

```
array([[-0.02884148,  0.06147919,  8.63946404,  0.87909706, -0.73320914,
         1.87747466, -0.79686079, -0.38642038, -0.54617318, -0.15474124,
        -0.05750978,  0.91308481, -0.15144388, -0.54415837, -0.15833061,
         0.30999031,  0.26940941,  0.12630598]])
```

important top 10 features

In [60]:

```
feature_names = X_train.columns
coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
top = coefs_with_fns[:-(10 + 1):-1]
for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s" % (coef_1, fn_1))
```

```
 8.6395 customer_ctr_score
 1.8775 customer_product_variation_score
 0.9131 X1_4
 0.8791 customer_stay_score
 0.3100 customer_active_segment_3
 0.2694 customer_active_segment_4
 0.1263 customer_active_segment_5
 0.0615 customer_product_search_score
 -0.0288 customer_visit_score
 -0.0575 X1_3
```

In [61]:

```
feature_names = X_train.columns
coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
top = coefs_with_fns[:10]
for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s" % (coef_1, fn_1))
```

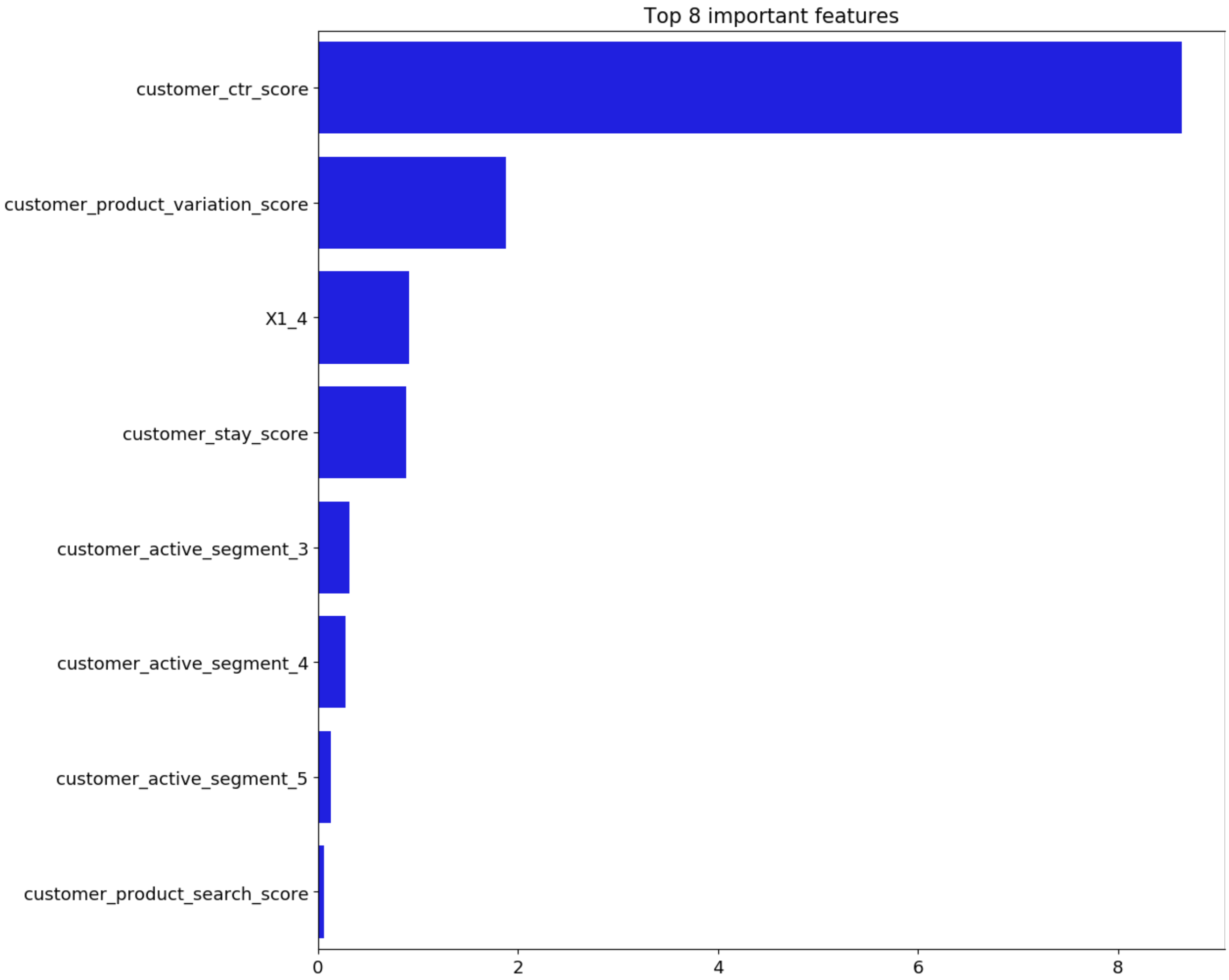
```
 -0.7969 customer_order_score
 -0.7332 customer_frequency_score
 -0.5462 X1_1
 -0.5442 customer_active_segment_1
 -0.3864 customer_affinity_score
 -0.1583 customer_active_segment_2
 -0.1547 X1_2
 -0.1514 X1_5
 -0.0575 X1_3
 -0.0288 customer_visit_score
```

In [74]:

```
coef=[]
fn=[]
for (coef_1, fn_1) in  coefs_with_fns[:-(8 + 1):-1]:
    coef.append(coef_1)
    fn.append(fn_1)
```

In [81]:

```
fig, axes = plt.subplots(ncols=1, figsize=(10, 10), dpi=100)
plt.tight_layout()

sns.barplot(y=fn, x=coef, ax=axes, color='blue')
#sns.barplot(y=df_nondisaster_unigrams[0].values[:N], x=df_nondisaster_unigrams[1].values[:N], ax=
axes[1], color='green')


axes.spines['right'].set_visible(False)
axes.set_xlabel('')
axes.set_ylabel('')
axes.tick_params(axis='x', labelsize=13)
axes.tick_params(axis='y', labelsize=13)

axes.set_title(f'Top {8} important features', fontsize=15)
#axes[1].set_title(f'Top {N} most common unigrams in Non-disaster Tweets', fontsize=15)

plt.show()
```



RBF kernal SVM

In [48]:

```
from sklearn.svm import SVC
C=[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000]
sv=SVC(kernel='rbf')
parameters = {'C':[0.001, 0.01, 0.1, 1, 10, 100, 1000]}
clf = GridSearchCV(sv, parameters, cv=10, scoring='average_precision',return_train_score=True)
clf.fit(X_train, y_train)
```

Out[48]:

```
GridSearchCV(cv=10, error_score=nan,
```

```
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight={0: 90, 1: 10}, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='average_precision', verbose=0)
```

In [49]:

```
clf.best_params_
```

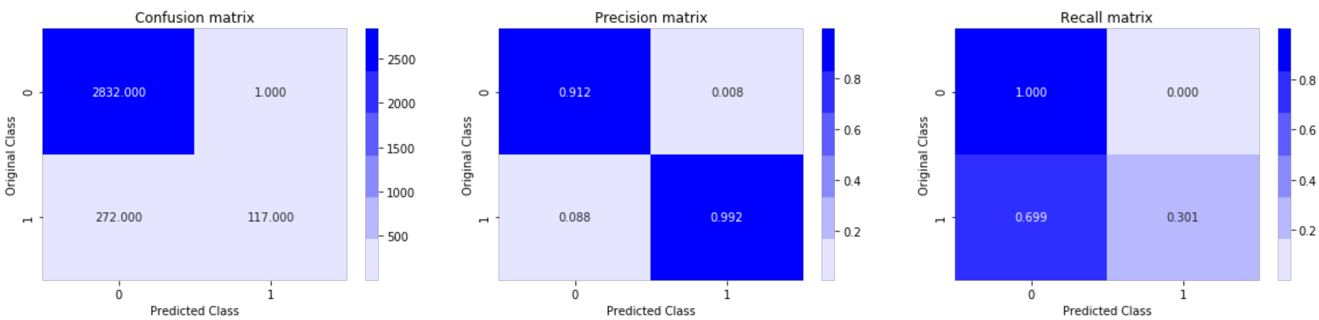
Out[49]:

```
{'C': 0.001}
```

In [50]:

```
predict_y = clf.predict(X_train)
print('For values of best C = ', 0.001 , "The train precision_score is:",precision_score(y_train, predict_y, average='macro'))
predict_y = clf.predict(X_test)
print('For values of best C = ', 0.001 , "The test precision_score is:",precision_score(y_test, predict_y, average='macro'))
#predicted_y =np.argmax(predict_y,axis=0)
print("Total number of data points :", len(predict_y))
plot_confusion_matrix(y_test, predict_y)
```

```
For values of best C =  0.001 The train precision_score is: 0.9522720070332547
For values of best alpha =  0.001 The test precision_score is: 0.951948278874716
Total number of data points : 3222
```



Random Forest

In [51]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=0,class_weight={0:10,1:90})
parameters = [{'n_estimators':[300,500,600,800,900],'max_depth':[4,6, 8, 9,10,12,14,17]}]
clf = GridSearchCV(rf, parameters, cv=4, scoring='average_precision',return_train_score=True)
clf.fit(X_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [52]:

```
clf.best_params_
```

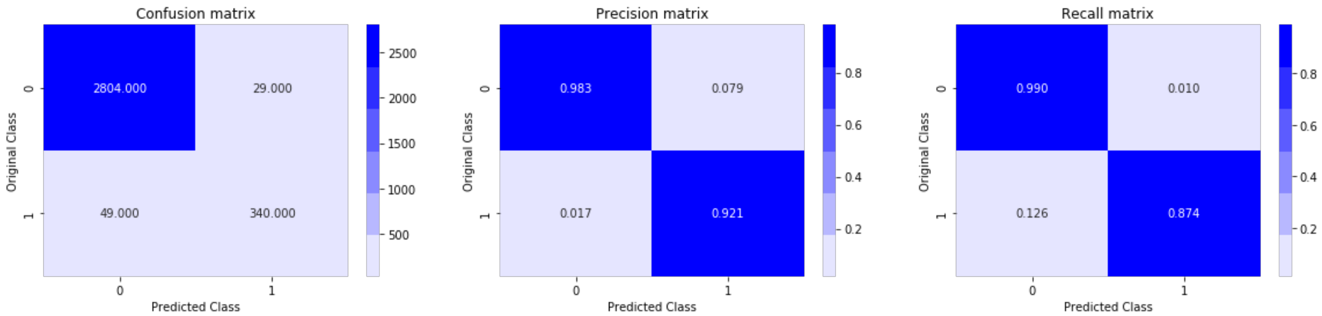
Out[52]:

```
{'max_depth': 12, 'n_estimators': 500}
```

In [53]:

```
predict_y = clf.predict(X_train)
print("The train precision_score is:",precision_score(y_train, predict_y, average='macro'))
predict_y = clf.predict(X_test)
print("The test precision_score is:",precision_score(y_test, predict_y, average='macro'))
#predicted_y =np.argmax(predict_y,axis=0)
print("Total number of data points :", len(predict_y))
plot_confusion_matrix(y_test, predict_y)
```

```
The train precision_score is: 0.9855692233225994
The test precision_score is: 0.9521171552409531
Total number of data points : 3222
```



XGBOOST

In [57]:

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier()
parameters = { 'max_depth': [6,8,10,15],'n_estimators': [8,10,15,20,25,50]}
clf = GridSearchCV(xgb_model, parameters, cv=4, scoring='average_precision',return_train_score=True
)
clf.fit(X_train, y_train)
```

Out[57]:

```
GridSearchCV(cv=4, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estim...
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=None, verbosity=None),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [6, 8, 10, 15],
                         'n_estimators': [8, 10, 15, 20, 25, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='average_precision', verbose=0)
```

In [58]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [59]:

```
clf.best_params_
```

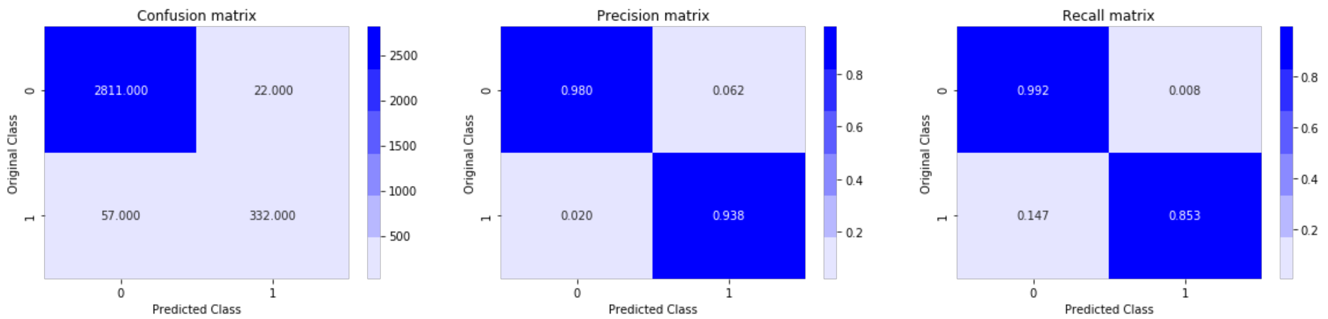
Out[59]:

```
{'max_depth': 6, 'n_estimators': 25}
```

In [60]:

```
predict_y = clf.predict(X_train)
print("The train precision_score is:",precision_score(y_train, predict_y, average='macro'))
predict_y = clf.predict(X_test)
print("The test precision_score is:",precision_score(y_test, predict_y, average='macro'))
#predicted_y =np.argmax(predict_y,axis=0)
print("Total number of data points :", len(predict_y))
plot_confusion_matrix(y_test, predict_y)
```

```
The train precision_score is: 0.9831424334965515
The test precision_score is: 0.9589893151785925
Total number of data points : 3222
```



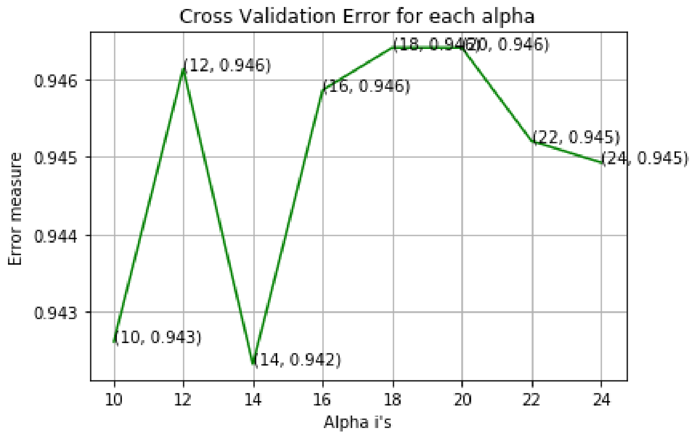
KNN

In [39]:

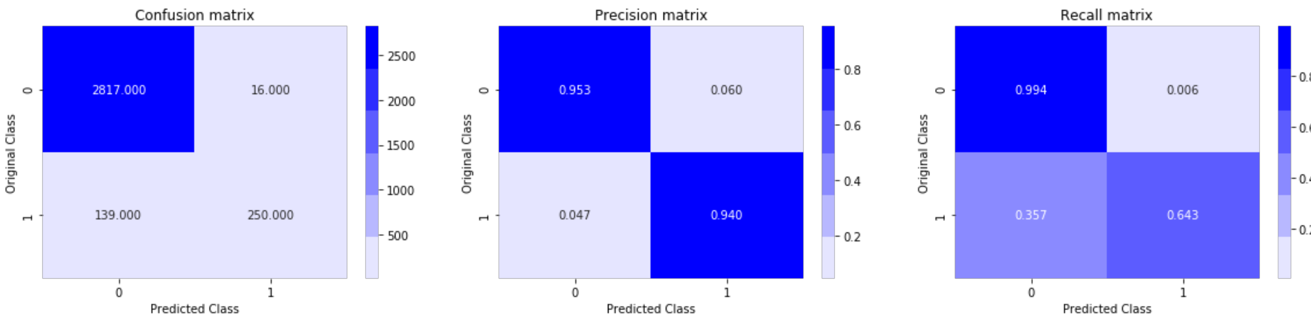
```
from sklearn.neighbors import KNeighborsClassifier
alpha=[10,12,14,16,18,20,22,24]
log_error_array=[]
for i in alpha:
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(X_train,y_train)
    predict_y = clf.predict(X_test)
    log_error_array.append(precision_score(y_test, predict_y, average='macro'))
    print('For values of k = ', i, "The precision_score is:",precision_score(y_test, predict_y, ave
rage='macro'))

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],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.argmax(log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(X_train, y_train)
#sig_clf = CalibratedClassifierCV(clf, method="sigmoid")


predict_y = clf.predict(X_train)
print(set(y_train) - set(predict_y))
print('For values of best alpha = ', alpha[best_alpha], "The train precision_score
is:",precision_score(y_train, predict_y, average='macro'))
predict_y = clf.predict(X_test)
```

```
print(set(y_test) - set(predict_y))
print('For values of best alpha = ', alpha[best_alpha], "The test precision_score is:",precision_score(y_test, predict_y, average='macro'))
#predicted_y =np.argmax(predict_y,axis=0)
print("Total number of data points :", len(predict_y))
plot_confusion_matrix(y_test, predict_y)
```

```
For values of k =  10 The precision_score is: 0.9426034715488887
For values of k =  12 The precision_score is: 0.9461386795643213
For values of k =  14 The precision_score is: 0.9423156165108305
For values of k =  16 The precision_score is: 0.9458632982973754
For values of k =  18 The precision_score is: 0.9464133100003052
For values of k =  20 The precision_score is: 0.9464133100003052
For values of k =  22 The precision_score is: 0.945204737920104
For values of k =  24 The precision_score is: 0.9449254236603037
```




```
set()
For values of best alpha =  18 The train precision_score is: 0.9447868849655611
set()
For values of best alpha =  18 The test precision_score is: 0.9464133100003052
Total number of data points : 3222
```



Conclusion

In [83]:

```
from prettytable import PrettyTable
```

In [84]:

```
pt=PrettyTable()
pt.field_names=["Architecture","Train macro precision","CV macro precision","Test precision"]
pt.add_row(["logistic regression","0.964","0.975","0.9638"])
pt.add_row(["RBF kernal SVM","0.952","0.951","0.91"])
pt.add_row(["Random forest","0.985","0.952","0.821"])
pt.add_row(["XGBOOST","0.983","0.958","0.89"])
print(pt)
```

```
+---------------------+-----------------------+--------------------+----------------+
|     Architecture    | Train macro precision | CV macro precision | Test precision |
```

```
+---------------------+-----------------------+--------------------+----------------+
| logistic regression |         0.964         |       0.975        |     0.9638     |
|    RBF kernal SVM   |         0.952         |       0.951        |      0.91      |
|    Random forest    |         0.985         |       0.952        |     0.821      |
|       XGBOOST       |         0.983         |       0.958        |      0.89      |
+---------------------+-----------------------+--------------------+----------------+
```

In [ ]: