In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc
from wordcloud import WordCloud, STOPWORDS
import re
import nltk
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

In [3]:
```
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = [10, 5]
warnings.filterwarnings("ignore", category=FutureWarning)
%config InlineBackend.figure_format = 'retina'
```

In [4]:
```
# avoid decoding problems
df = pd.read_csv("train.csv")

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ---------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
# ---------------- python 3 ---------------------
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [5]:
```
df.head()
```

Out[5]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [6]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
```

In [7]:
```
# To get the results in 4 decemal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("’", "'").replace("
’", "'")\
                           .replace("won't", "will not").replace("cannot", "can not
").replace("can't", "can not")\
                           .replace("n't", " not").replace("what's", "what is").rep
lace("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace(
"'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").rep
lace("'s", " own")\
                           .replace("%", " percent ").replace("₹", " rupee ").repla
ce("$", " dollar ")\
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


    porter = PorterStemmer()
    pattern = re.compile('\W')

    if type(x) == type(''):
        x = re.sub(pattern, ' ', x)


    if type(x) == type(''):
        x = porter.stem(x)
        example1 = BeautifulSoup(x)
        x = example1.get_text()


    return x
```

In [8]:
```
def get_token_features(q1, q2):
    token_features = [0.0]*10

    # Converting the Sentence into Tokens:
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])

    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])

    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))

    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))

    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SA
FE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SA
FE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SA
FE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SA
FE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) +
SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) +
SAFE_DIV)

    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

    # First word of both question is same or not
    token_features[7] = int(q1_tokens[0] == q2_tokens[0])

    token_features[8] = abs(len(q1_tokens) - len(q2_tokens))

    #Average Token Length of both Questions
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

def extract_features(df):
    # preprocessing each question
    df["question1"] = df["question1"].fillna("").apply(preprocess)
```

In [9]:
```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[9]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983	...	0.78
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988	...	0.46

2 rows × 21 columns

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(ngram_range=(1,4))
df_q_one = vect.fit_transform(df['question1'].values.astype('U'))
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-value
error-np-nan-is-an-invalid-document
df_q_one.shape
```

Out[10]: (404290, 4068576)

ooook, this would take a lot of time

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(ngram_range=(1,4),max_features=20000)
df_q_one = vect.fit_transform(df['question1'].values.astype('U'))
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-value
error-np-nan-is-an-invalid-document
df_q_one.shape
```

Out[10]: (404290, 20000)

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(ngram_range=(1,4),max_features=20000)
df_q_two = vect.fit_transform(df['question2'].values.astype('U'))

df_q_two.shape
```

Out[11]: (404290, 20000)

In [12]:
```
X = df.drop(['is_duplicate'],axis=1)
```

In [13]:
```
import scipy.sparse
from scipy.sparse import coo_matrix, hstack
X = scipy.sparse.hstack((df_q_one,df_q_two))
```

In [14]:
```
X
```

Out[14]:
```
<404290x40000 sparse matrix of type '<class 'numpy.float64'>'
        with 13720667 stored elements in COOrdinate format>
```

In [15]:
```
y_true = df['is_duplicate']
from sklearn.cross_validation import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X, y_true, stratify=y_true, test
_size=0.2)
```

In [16]:
```
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 : (323432, 40000)
Number of data points in test data : (80858, 40000)
```

In [21]:
```
from collections import Counter
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])/tr
ain_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)
```

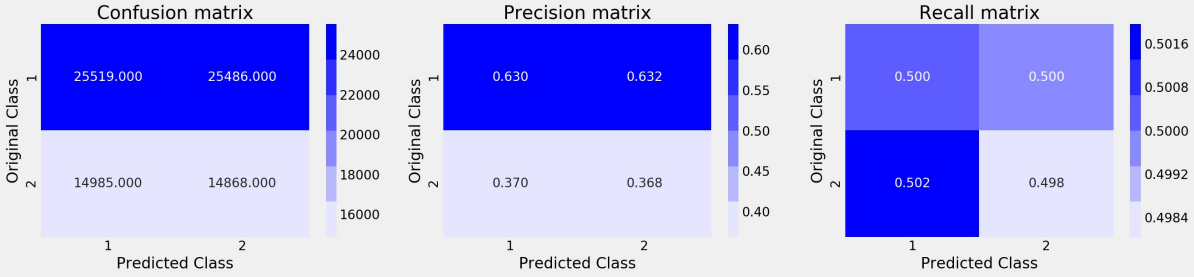
```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308033837097133 Class 1:  0.36919661629028666
---------- Distribution of output variable in test data ----------
Class 0:  0.36920279997031835 Class 1:  0.36920279997031835
```

In [18]:
```
# 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 pr
edicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divide each element of the confusion matrix with the sum of elements in that c
olumn

    # 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 dimensionsl 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 ro
w
    # 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 = [1,2]
    # 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, yticklabel
s=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, yticklabel
s=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, yticklabel
s=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

Building a random model so that we can find out the threshold of our log loss.

In [20]:
```
# 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
# we create a output array that has exactly same size as the CV data
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.metrics import confusion_matrix
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    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, eps=
1e-15))

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

```
Log loss on Test Data using Random Model 0.8905656559613436
```

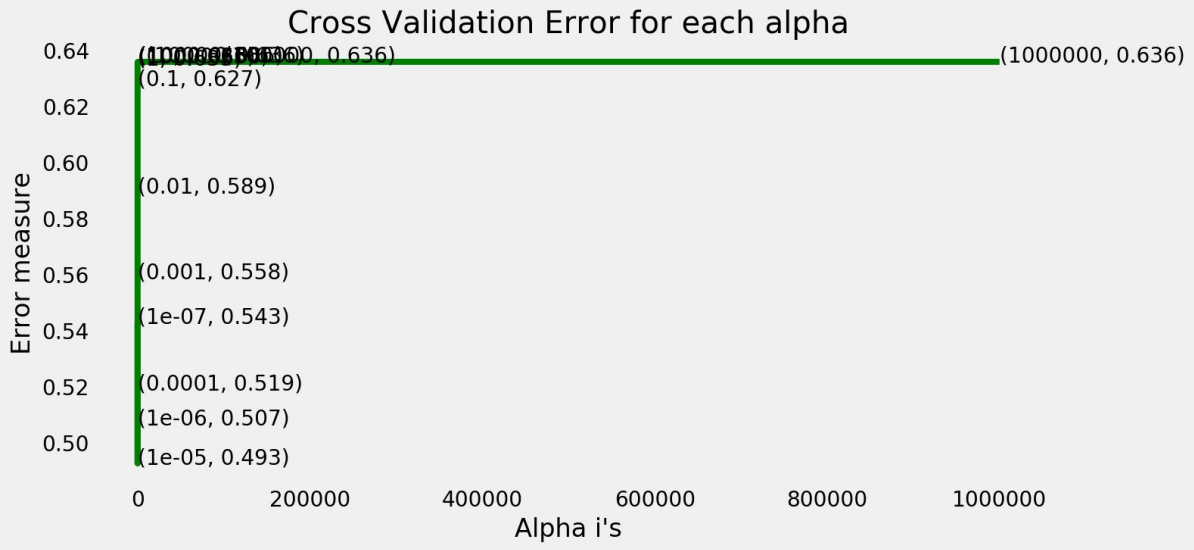


Logistic Regression using SGD classifier with log loss

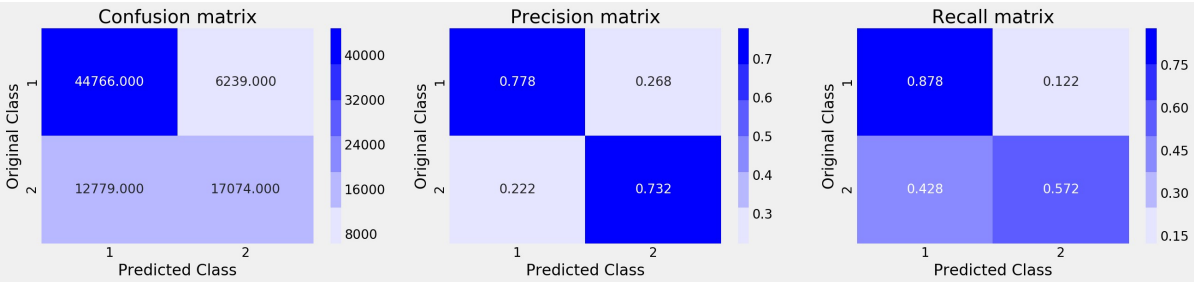
In [19]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
import scipy.stats as st
from datetime import datetime
from sklearn.linear_model import LogisticRegression
```

In [22]:
```
start = datetime.now()


alpha = [10 ** x for x in range(-7, 7)] # hyperparam for SGD classifier.




log_error_array=[]
for i in alpha:
    clf = SGDClassifier(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)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-
15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_
y, labels=clf.classes_, eps=1e-15))

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.argmin(log_error_array)
clf = SGDClassifier(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:",lo
g_loss(y_train, 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))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)


print('='*100)
print("Time taken to run this cell :", datetime.now() - start)
print('='*100)
```

```
For values of alpha =  1e-07 The log loss is: 0.5427532877913069
For values of alpha =  1e-06 The log loss is: 0.5065490234258635
For values of alpha =  1e-05 The log loss is: 0.4925248740364623
For values of alpha =  0.0001 The log loss is: 0.5189720286680636
For values of alpha =  0.001 The log loss is: 0.5584789989290293
For values of alpha =  0.01 The log loss is: 0.5890861957952741
For values of alpha =  0.1 The log loss is: 0.627487140629177
For values of alpha =  1 The log loss is: 0.6348418859884559
For values of alpha =  10 The log loss is: 0.635580287554288
For values of alpha =  100 The log loss is: 0.6356707890458261
For values of alpha =  1000 The log loss is: 0.6356888704374231
For values of alpha =  10000 The log loss is: 0.6356957061890495
For values of alpha =  100000 The log loss is: 0.6356992066934917
For values of alpha =  1000000 The log loss is: 0.6357011376974551
```




```
For values of best alpha =  1e-05 The train log loss is: 0.461575676304871
For values of best alpha =  1e-05 The test log loss is: 0.4925248740364623
Total number of data points : 80858
```

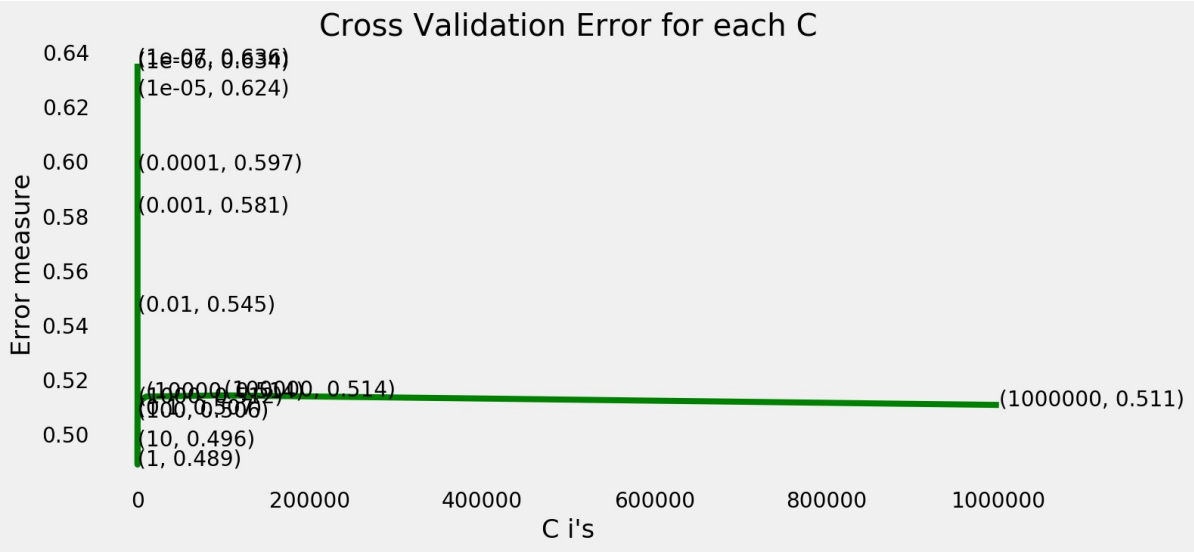



```
====================================================================================================
Time taken to run this cell : 0:00:57.692394
====================================================================================================
```

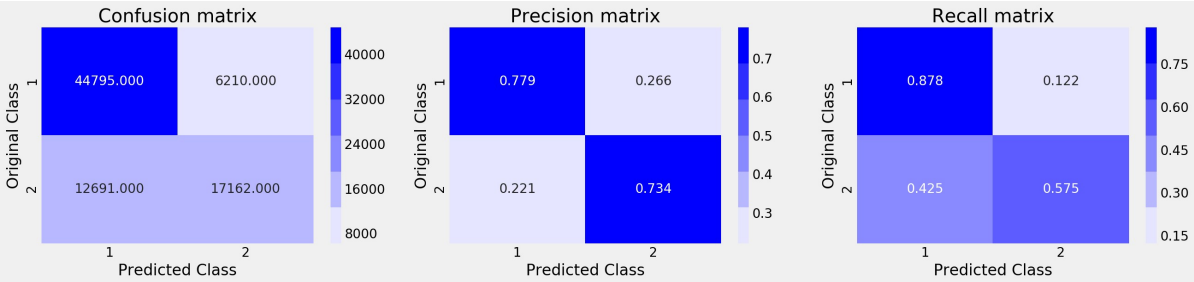
Logistic Regression : Logistic Regression

In [23]:
```
start = datetime.now()


C = [10 ** x for x in range(-7, 7)] # hyperparam for linear regression classifier.




log_error_array=[]
for i in C:
    clf = LogisticRegression(C = i, penalty='l2', 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_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-
15))
    print('For values of C = ', i, "The log loss is:",log_loss(y_test, predict_y, l
abels=clf.classes_, eps=1e-15))

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


best_C = np.argmin(log_error_array)
clf = LogisticRegression(C = C[best_C], penalty='l2', 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 C = ', C[best_C], "The train log loss is:",log_loss(y_tra
in, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best C = ', C[best_C], "The test log loss is:",log_loss(y_test
, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)


print('='*100)
print("Time taken to run this cell :", datetime.now() - start)
print('='*100)
```

```
For values of C =  1e-07 The log loss is: 0.6355753198436918
For values of C =  1e-06 The log loss is: 0.6344302957246559
For values of C =  1e-05 The log loss is: 0.6242502468212713
For values of C =  0.0001 The log loss is: 0.5970285048148198
For values of C =  0.001 The log loss is: 0.5812780461459526
For values of C =  0.01 The log loss is: 0.5450492336812047
For values of C =  0.1 The log loss is: 0.5073508279446839
For values of C =  1 The log loss is: 0.4887340804280247
For values of C =  10 The log loss is: 0.49579327729538286
For values of C =  100 The log loss is: 0.5062496907263982
For values of C =  1000 The log loss is: 0.5116333453758863
For values of C =  10000 The log loss is: 0.5136546268594419
For values of C =  100000 The log loss is: 0.5140976458355204
For values of C =  1000000 The log loss is: 0.5105868284714733
```




```
For values of best C =  1 The train log loss is: 0.448829242134321
For values of best C =  1 The test log loss is: 0.4887340804280247
Total number of data points : 80858
```

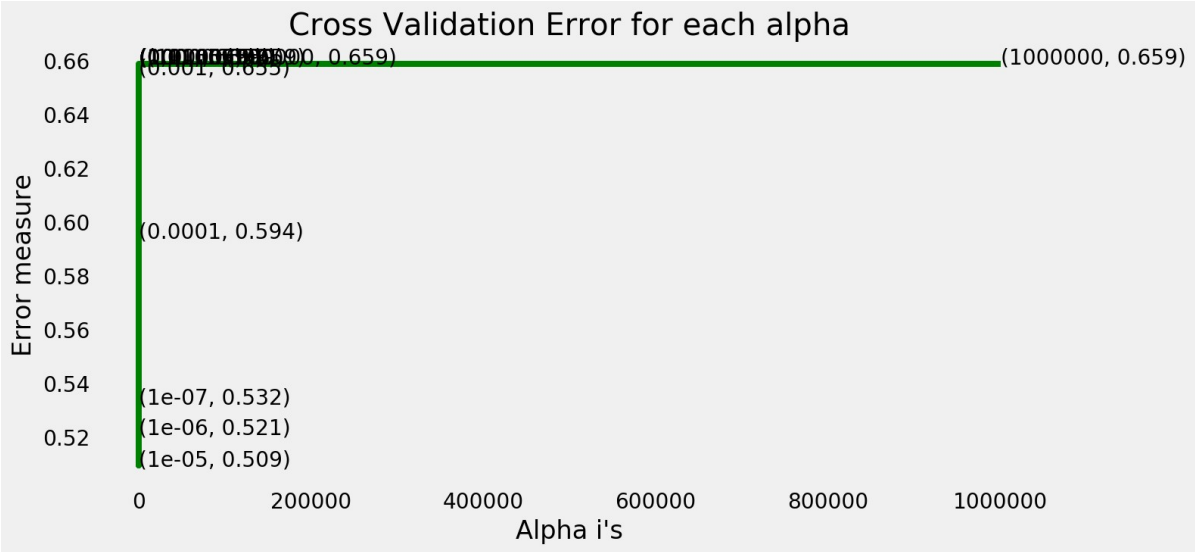



```
==========================================================================================
Time taken to run this cell : 1:03:28.628900
==========================================================================================
```

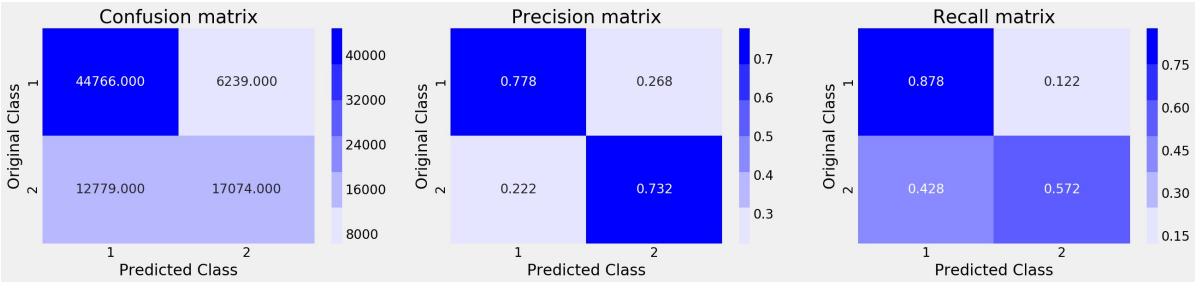
Linear SVM

In [24]:
```
start = datetime.now()


alpha = [10 ** x for x in range(-7, 7)] # hyperparam for SGD classifier.

#
#------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', 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_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-
15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_
y, labels=clf.classes_, eps=1e-15))

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.argmin(log_error_array)
clf = SGDClassifier(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:",lo
g_loss(y_train, 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))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)


print('='*100)
print("Time taken to run this cell :", datetime.now() - start)
print('='*100)
```

```
For values of alpha =  1e-07 The log loss is: 0.5323382145982767
For values of alpha =  1e-06 The log loss is: 0.5208668162887481
For values of alpha =  1e-05 The log loss is: 0.5092306475390 56
For values of alpha =  0.0001 The log loss is: 0.5938705654286839
For values of alpha =  0.001 The log loss is: 0.6546532352782019
For values of alpha =  0.01 The log loss is: 0.6585300338918999
For values of alpha =  0.1 The log loss is: 0.6585300338918075
For values of alpha =  1 The log loss is: 0.6585300338919
For values of alpha =  10 The log loss is: 0.6585300338918159
For values of alpha =  100 The log loss is: 0.6585300338919
For values of alpha =  1000 The log loss is: 0.6585300338918629
For values of alpha =  10000 The log loss is: 0.6585300338918157
For values of alpha =  100000 The log loss is: 0.6585300338919
For values of alpha =  1000000 The log loss is: 0.6585300338918667
```




```
For values of best alpha =  1e-05 The train log loss is: 0.461575676304871
For values of best alpha =  1e-05 The test log loss is: 0.4925248740364623
Total number of data points : 80858
```




```
==========================================================================================
Time taken to run this cell : 0:01:06.520736
==========================================================================================
```

In [21]:
```
start = datetime.now()

#https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/


param = {
        'silent': [False],
        'max_depth': [2,4,6,8,10],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3,0.5,0.75,1],
        'subsample': [0.001,0.01,0.1,0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bytree': [0.2,0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bylevel': [0.2,0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
        'gamma': [0, 0.25, 0.5,0.75, 1.0],
        'reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
        'reg_alpha':[1e-5, 1e-2, 0.1, 1, 10, 100],
        'n_estimators': [1000]}



fit_params = {'eval_metric': 'logloss',
              'early_stopping_rounds': 10,
              'eval_set': [(X_train, y_train)]
              }

clf = xgb.XGBClassifier(nthread = 4)


model = RandomizedSearchCV(clf, param, n_iter=20,
                           n_jobs=1, verbose=2, cv=2,
                           fit_params=fit_params,
                           scoring='neg_log_loss', refit=False, random_state=42)


model.fit(X_train, y_train)


print('='*100)
print("Time taken to run this cell :", datetime.now() - start)
print('='*100)
```

```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
[CV] subsample=1.0, silent=False, reg_lambda=50.0, reg_alpha=10, n_estimators=10
00, min_child_weight=3.0, max_depth=10, learning_rate=0.1, gamma=0.75, colsample
_bytree=0.6, colsample_bylevel=1.0
[21:43:53] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 280 extra nodes, 60 pruned nodes, max_depth=10
[0]     validation_0-logloss:0.679089
Will train until validation_0-logloss hasn't improved in 10 rounds.
[21:43:54] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 246 extra nodes, 62 pruned nodes, max_depth=10
[1]     validation_0-logloss:0.667376
[21:43:55] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 308 extra nodes, 66 pruned nodes, max_depth=10
[2]     validation_0-logloss:0.656798
[21:43:56] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 272 extra nodes, 50 pruned nodes, max_depth=10
[3]     validation_0-logloss:0.649069
[21:43:57] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 272 extra nodes, 52 pruned nodes, max_depth=10
[4]     validation_0-logloss:0.641106
[21:43:57] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 224 extra nodes, 34 pruned nodes, max_depth=10
[5]     validation_0-logloss:0.634692
[21:43:58] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 136 extra nodes, 30 pruned nodes, max_depth=10
[6]     validation_0-logloss:0.62995
[21:43:59] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 264 extra nodes, 58 pruned nodes, max_depth=10
[7]     validation_0-logloss:0.625115
[21:44:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 234 extra nodes, 50 pruned nodes, max_depth=10
[8]     validation_0-logloss:0.620508
[21:44:01] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 246 extra nodes, 30 pruned nodes, max_depth=10
[9]     validation_0-logloss:0.61661
[21:44:01] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 170 extra nodes, 50 pruned nodes, max_depth=10
[10]    validation_0-logloss:0.613459
[21:44:02] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 246 extra nodes, 36 pruned nodes, max_depth=10
[11]    validation_0-logloss:0.609959
[21:44:03] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 162 extra nodes, 58 pruned nodes, max_depth=10
[12]    validation_0-logloss:0.60723
[21:44:04] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 166 extra nodes, 66 pruned nodes, max_depth=10
[13]    validation_0-logloss:0.60482
[21:44:05] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 288 extra nodes, 72 pruned nodes, max_depth=10
[14]    validation_0-logloss:0.601668
[21:44:05] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 148 extra nodes, 24 pruned nodes, max_depth=10
[15]    validation_0-logloss:0.599305
[21:44:06] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 224 extra nodes, 60 pruned nodes, max_depth=10
[16]    validation_0-logloss:0.59703
[21:44:07] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 168 extra nodes, 36 pruned nodes, max_depth=10
[17]    validation_0-logloss:0.59518
[21:44:08] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 226 extra nodes, 68 pruned nodes, max_depth=10
[18]    validation_0-logloss:0.593005
[21:44:09] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 226 extra nodes, 86 pruned nodes, max_depth=10
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 12.8min remaining:    0.0s
```

```
[21:56:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 288 extra nodes, 54 pruned nodes, max_depth=10
[0]     validation_0-logloss:0.678996
Will train until validation_0-logloss hasn't improved in 10 rounds.
[21:56:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 230 extra nodes, 54 pruned nodes, max_depth=10
[1]     validation_0-logloss:0.667265
[21:56:44] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 284 extra nodes, 56 pruned nodes, max_depth=10
[2]     validation_0-logloss:0.657817
[21:56:45] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 270 extra nodes, 90 pruned nodes, max_depth=10
[3]     validation_0-logloss:0.650021
[21:56:45] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 286 extra nodes, 84 pruned nodes, max_depth=10
[4]     validation_0-logloss:0.642373
[21:56:46] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 274 extra nodes, 68 pruned nodes, max_depth=10
[5]     validation_0-logloss:0.636409
[21:56:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 204 extra nodes, 44 pruned nodes, max_depth=10
[6]     validation_0-logloss:0.631165
[21:56:48] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 236 extra nodes, 38 pruned nodes, max_depth=10
[7]     validation_0-logloss:0.626478
[21:56:49] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 192 extra nodes, 36 pruned nodes, max_depth=10
[8]     validation_0-logloss:0.62232
[21:56:49] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 268 extra nodes, 50 pruned nodes, max_depth=10
[9]     validation_0-logloss:0.618192
[21:56:50] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 236 extra nodes, 42 pruned nodes, max_depth=10
[10]    validation_0-logloss:0.614535
[21:56:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 208 extra nodes, 54 pruned nodes, max_depth=10
[11]    validation_0-logloss:0.611304
[21:56:52] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 156 extra nodes, 58 pruned nodes, max_depth=10
[12]    validation_0-logloss:0.608702
[21:56:52] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 154 extra nodes, 46 pruned nodes, max_depth=10
[13]    validation_0-logloss:0.606173
[21:56:53] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 254 extra nodes, 34 pruned nodes, max_depth=10
[14]    validation_0-logloss:0.603045
[21:56:54] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 226 extra nodes, 50 pruned nodes, max_depth=10
[15]    validation_0-logloss:0.600823
[21:56:55] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 272 extra nodes, 46 pruned nodes, max_depth=10
[16]    validation_0-logloss:0.59787
[21:56:56] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 232 extra nodes, 74 pruned nodes, max_depth=10
[17]    validation_0-logloss:0.59554
[21:56:56] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 244 extra nodes, 52 pruned nodes, max_depth=10
[18]    validation_0-logloss:0.593371
[21:56:57] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 232 extra nodes, 76 pruned nodes, max_depth=10
[19]    validation_0-logloss:0.591093
[21:56:58] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 214 extra nodes, 24 pruned nodes, max_depth=10
[20]    validation_0-logloss:0.589313
```

```
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed: 256.1min finished
```

In [22]:
```
best_score = model.best_score_
best_params = model.best_params_

print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: -0.4553916776458775
Best params:
colsample_bylevel: 0.7
colsample_bytree: 0.8
gamma: 0.5
learning_rate: 0.3
max_depth: 8
min_child_weight: 1.0
n_estimators: 1000
reg_alpha: 1e-05
reg_lambda: 10.0
silent: False
subsample: 0.5
```

In [20]:
```
start = datetime.now()

clf = xgb.XGBClassifier(max_depth = 8,n_estimators = 1000,learning_rate=0.3,objecti
ve='binary:logistic',min_child_weight=1.0,
                        reg_alpha = 1e-05, colsample_bytree = 0.8,colsample_bylevel
= 0.7, gamma = 0.5, reg_lambda = 10, subsample = 0.5,
                        silent = False)

clf_2 = CalibratedClassifierCV(clf, method="sigmoid")
clf_2.fit(X_train, y_train)
predict_y = clf_2.predict_proba(X_test)
log_loss(y_test, predict_y, eps=1e-15)
print("The log loss is:",log_loss(y_test, predict_y,  eps=1e-15))

print('='*100)
print("Time taken to run this cell :", datetime.now() - start)
print('='*100)
```

```
[13:09:10] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 182 extra nodes, 10 pruned nodes, max_depth=8
[13:09:12] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 152 extra nodes, 12 pruned nodes, max_depth=8
[13:09:13] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 156 extra nodes, 6 pruned nodes, max_depth=8
[13:09:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 214 extra nodes, 18 pruned nodes, max_depth=8
[13:09:17] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 218 extra nodes, 10 pruned nodes, max_depth=8
[13:09:19] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 194 extra nodes, 2 pruned nodes, max_depth=8
[13:09:20] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 188 extra nodes, 20 pruned nodes, max_depth=8
[13:09:22] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 266 extra nodes, 22 pruned nodes, max_depth=8
[13:09:23] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 184 extra nodes, 24 pruned nodes, max_depth=8
[13:09:25] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 134 extra nodes, 4 pruned nodes, max_depth=8
[13:09:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 154 extra nodes, 18 pruned nodes, max_depth=8
[13:09:28] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 204 extra nodes, 12 pruned nodes, max_depth=8
[13:09:29] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 110 extra nodes, 6 pruned nodes, max_depth=8
[13:09:31] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 194 extra nodes, 28 pruned nodes, max_depth=8
[13:09:32] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 130 extra nodes, 14 pruned nodes, max_depth=8
[13:09:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 190 extra nodes, 24 pruned nodes, max_depth=8
[13:09:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 210 extra nodes, 22 pruned nodes, max_depth=8
[13:09:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 188 extra nodes, 14 pruned nodes, max_depth=8
[13:09:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 120 extra nodes, 10 pruned nodes, max_depth=8
[13:09:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 126 extra nodes, 14 pruned nodes, max_depth=8
[13:09:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 140 extra nodes, 8 pruned nodes, max_depth=8
[13:09:44] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 108 extra nodes, 6 pruned nodes, max_depth=8
[13:09:45] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 114 extra nodes, 16 pruned nodes, max_depth=8
[13:09:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 126 extra nodes, 16 pruned nodes, max_depth=8
[13:09:48] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 130 extra nodes, 14 pruned nodes, max_depth=8
[13:09:50] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 108 extra nodes, 6 pruned nodes, max_depth=8
[13:09:52] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 192 extra nodes, 20 pruned nodes, max_depth=8
[13:09:53] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 112 extra nodes, 12 pruned nodes, max_depth=8
[13:09:54] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 116 extra nodes, 12 pruned nodes, max_depth=8
[13:09:56] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 152 extra nodes, 12 pruned nodes, max_depth=8
[13:09:57] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 112 extra nodes, 10 pruned nodes, max_depth=8
[13:09:59] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74:
tree pruning end, 1 roots, 114 extra nodes, 10 pruned nodes, max_depth=8
```

Here the test log loss is 0.429858415380801 which is pretty good considering our previous models

In [24]:
```
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
Total number of data points : 80858
```