In [1]:

```
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 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
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [2]:

```
#Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('final_features.csv', names=['Unnamed: 0','id','is_duplicate','cw
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
```

In [12]:

```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None


def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the databse:")
    tables =table_names.fetchall()
    print(tables[0][0])
    return(len(tables))
```

In [4]:

```
read_db = 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
data
```

In [5]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)

        # for selecting random points
        data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn
        conn_r.commit()
        conn_r.close()
```

In [6]:

```
# remove the first row
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace=True)
```

In [7]:

```
data.head()
```

Out[7]:

	cwc_min	cwc_max	csc_min	csc_max	ctc
1	0.499975001249937	0.249993750156246	0.249993750156246	0.166663888935184	0.2499968750
2	0.799984000319994	0.666655555740738	0.799984000319994	0.666655555740738	0.7999920000
3	0.66664444518516	0.66664444518516	0.499975001249937	0.199996000079998	0.5999880002
4	0.833319444675922	0.555549382784636	0.999966667777741	0.374995312558593	0.8888790124
5	0.499987500312492	0.399992000159997	0.399992000159997	0.399992000159997	0.4444395062

5 rows × 218 columns

4.2 Converting strings to numerics

In [8]:

```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
cols = list(data.columns)
for i in cols:
    data[i] = data[i].apply(pd.to_numeric)
    print(i)
```

```
word_Total
word_share
freq_q1+q2
freq_q1-q2
0_x
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
10_x
11_x
12_x
13_x
14_x
15_x
```

In [2]:

```
df_nlp = pd.read_csv("nlp_features_train.csv", encoding = "latin-1")
df_nlp = df_nlp.fillna('')
df_nlp.head(2)
```

Out[2]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
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
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

2 rows × 21 columns

In [3]:

```
df_feat = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding = "latin-1")
df_feat = df_feat.fillna('')
df_feat.head(2)
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n
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	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	

In [4]:

```
df_total = pd.merge(df_nlp, df_feat, on = "id")
df_total.head(2)
```

Out[4]:

	id	qid1_x	qid2_x	question1_x	question2_x	is_duplicate_x	cwc_min	cwc_max	csc_min
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
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

2 rows × 37 columns

In [5]:

```
Y = df_total["is_duplicate_x"]
X= df_total.drop("is_duplicate_x", axis = 1)
```

4.3 Random train test split(70:30)

In [6]:

```
X_train,X_test, y_train, y_test = train_test_split(X,Y, stratify=Y, test_size=0.3)
```

In [7]:

```
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 : (283003, 36)
Number of data points in test data : (121287, 36)
```

In [8]:

```
X_train.columns
```

Out[8]:

```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_word_e
q',
       'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio',
       'token_sort_ratio', 'fuzz_ratio', 'fuzz_partial_ratio',
       'longest_substr_ratio', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', 'freq_qid1', 'freq_qid2', 'q1len',
       'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Total',
       'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

In [9]:

```
questions = np.array(X_train['question1_x'].astype("unicode")) + np.array(X_train['questior

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2))
vectorizer.fit(questions)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_))
vectorizer.idf_
vectorizer.get_feature_names()
```

```
 '04 lts',
 '06',
 '09',
 '10',
 '10 10',
 '10 11',
 '10 12',
 '10 15',
 '10 20',
 '10 am',
 '10 and',
 '10 best',
 '10 bollywood',
 '10 books',
 '10 can',
 '10 cbse',

 '10 cgpa',
 '10 class',
 '10 day'
```

In [12]:

```
pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.2.
```

```
Collecting https://github.com/explosion/spacy-models/releases/download/en_co
re_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.gz (https://github.com/explosion/sp
acy-models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.g
z)
  Downloading https://github.com/explosion/spacy-models/releases/download/en
_core_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.gz (https://github.com/explosio
n/spacy-models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.t
ar.gz) (12.0MB)
    100% |████████████████████████████████| 12.0MB 116kB/s eta 0:00:011
Building wheels for collected packages: en-core-web-sm
  Running setup.py bdist_wheel for en-core-web-sm ... done
  Stored in directory: /home/shanud6711/.cache/pip/wheels/6a/47/fb/6b5a0b890
6d8e8779246c67d4658fd8a544d4a03a75520197a
Successfully built en-core-web-sm
Installing collected packages: en-core-web-sm
Successfully installed en-core-web-sm-2.2.5
Note: you may need to restart the kernel to use updated packages.
```

In [10]:

```
import spacy
from tqdm import tqdm
import en_core_web_sm
# compute average word2vec for each review.
def avg_w2v_vectors(preprocessed_questions):
    nlp = en_core_web_sm.load()

    vecs1 = []
    # https://github.com/noamraph/tqdm
    # tqdm is used to print the progress bar
    for qu1 in tqdm(list(preprocessed_questions)):
        doc1 = nlp(qu1)


        # 384 is the number of dimensions of vectors
        mean_vec1 = np.zeros([len(doc1),96])
        for word1 in doc1:
            # word2vec
            vec1 = word1.vector
            # fetch df score
            try:
                idf = word2tfidf[str(word1)]
            except:
                idf = 0
            # compute final vec
            mean_vec1 += vec1 * idf
        mean_vec1 = mean_vec1.mean(axis=0)
        vecs1.append(mean_vec1)
    return vecs1
```

In [11]:

```
X_train_question1_w2v=avg_w2v_vectors(X_train['question1_x'])
X_test_question1_w2v=avg_w2v_vectors(X_test['question1_x'])

X_train_question2_w2v=avg_w2v_vectors(X_train['question2_x'])
X_test_question2_w2v=avg_w2v_vectors(X_test['question2_x'])
```

```
100%|██████████| 283003/283003 [45:23<00:00, 103.89it/s]
100%|██████████| 121287/121287 [20:14<00:00, 99.84it/s]
100%|██████████| 283003/283003 [47:59<00:00, 98.27it/s]
100%|██████████| 121287/121287 [20:55<00:00, 96.61it/s]
```

In [12]:

```
train_feats_w2v = X_train.drop(["id", "is_duplicate_y", "qid1_x", "qid2_x", "qid1_y", "qid2
train_feats_w2v.shape
```

Out[12]:

```
(283003, 26)
```

In [13]:

```
test_feats_w2v = X_test.drop(["id", "is_duplicate_y", "qid1_x", "qid2_x", "qid1_y", "qid2_y
test_feats_w2v.shape
```

Out[13]:

```
(121287, 26)
```

In [14]:

```
from scipy.sparse import hstack

X_train_w2v = hstack([train_feats_w2v,X_train_question1_w2v,X_train_question2_w2v],format =
X_test_w2v = hstack([test_feats_w2v,X_test_question1_w2v,X_test_question2_w2v],format = "cs
```

In [15]:

```
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:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in test data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

In [16]:

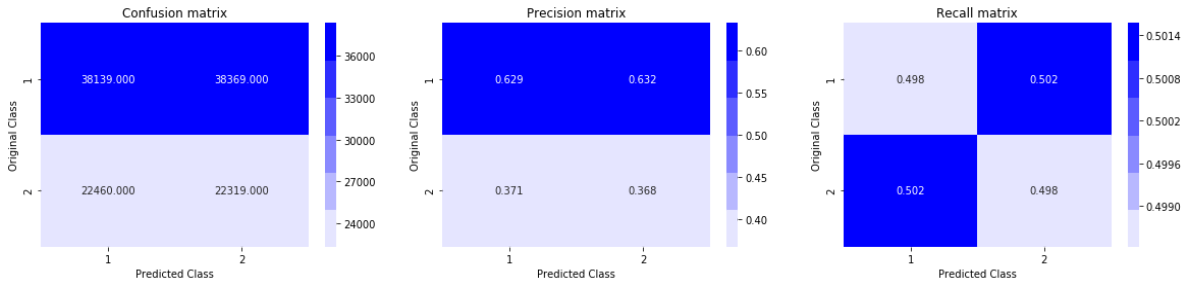
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted

    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 a
    # 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 a
    # 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, 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()
```

4.4 Building a random model (Finding worst-case log-loss)

In [50]:

```
# 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
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 0.8872038095555621
```



4.4 Logistic Regression with hyperparameter tuning

In [18]:

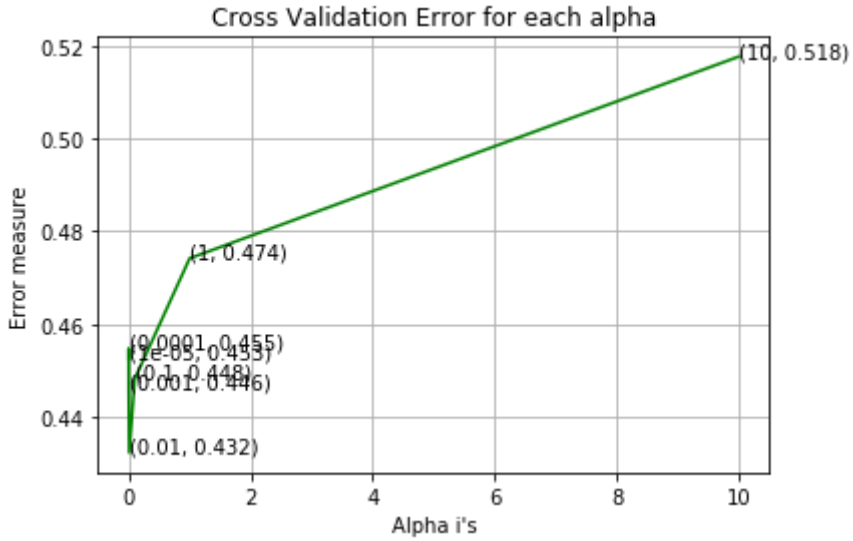
```
X_train_w2v=pd.SparseDataFrame(X_train_w2v).fillna(0)
X_test_w2v=pd.SparseDataFrame(X_test_w2v).fillna(0)
```

In [55]:

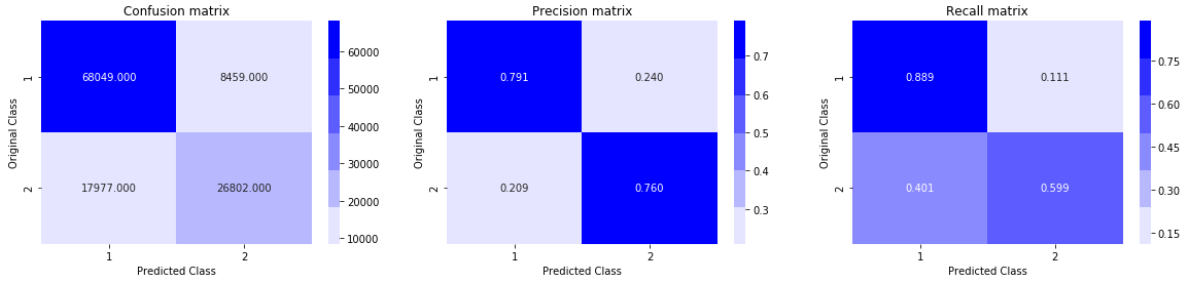
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Desce
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link:
#-------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, n_jobs = -1,cla
    clf.fit(X_train_w2v,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_w2v, y_train)
    predict_y = sig_clf.predict_proba(X_test_w2v)
    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, label

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_w2v, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_w2v, y_train)

predict_y = sig_clf.predict_proba(X_train_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(X_test_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.4525799176717347
For values of alpha =  0.0001 The log loss is: 0.4547417063157503
For values of alpha =  0.001 The log loss is: 0.4463228411402859
For values of alpha =  0.01 The log loss is: 0.43230362680216367
```

```
For values of alpha =  0.1 The log loss is: 0.44818391579502026
For values of alpha =  1 The log loss is: 0.47416026794108496
For values of alpha =  10 The log loss is: 0.5176170009417833
```




```
For values of best alpha =  0.01 The train log loss is: 0.4377011353634119
For values of best alpha =  0.01 The test log loss is: 0.43785816127243876
Total number of data points : 121287
```



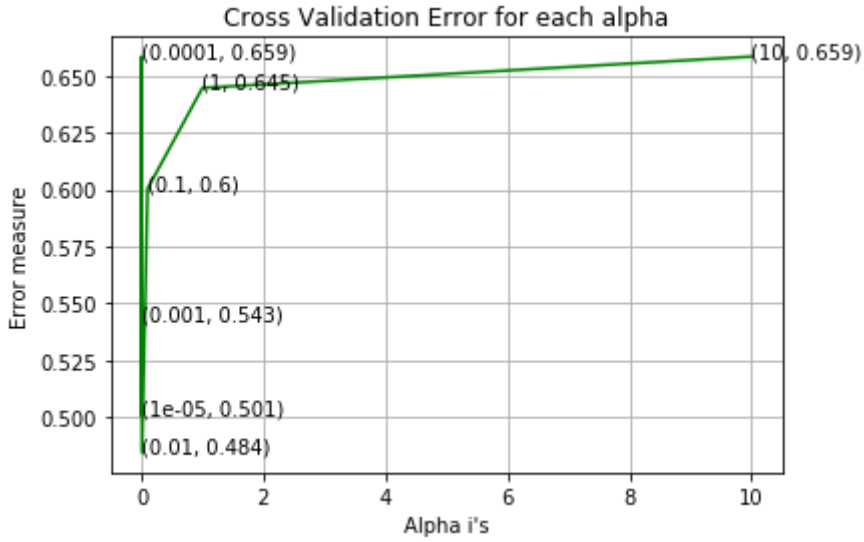
4.5 Linear SVM with hyperparameter tuning

In [20]:

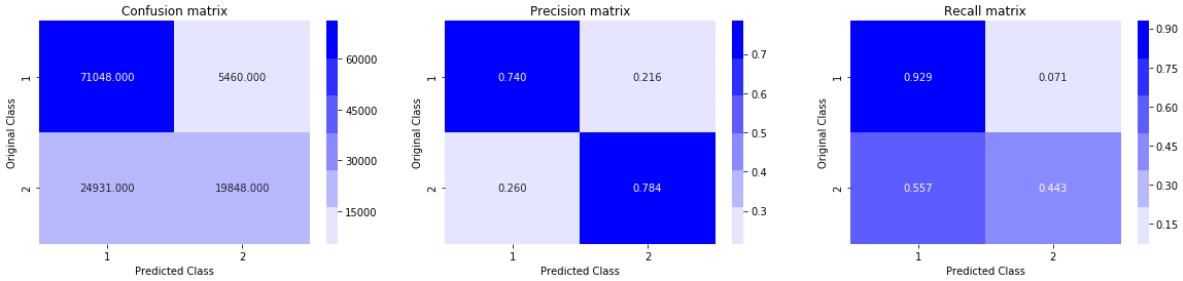
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Desce
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link:
#-------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42,n_jobs=-1)
    clf.fit(X_train_w2v, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_w2v, y_train)
    predict_y = sig_clf.predict_proba(X_test_w2v)
    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, label

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='l1', loss='hinge', random_state=42)
clf.fit(X_train_w2v, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_w2v, y_train)

predict_y = sig_clf.predict_proba(X_train_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(X_test_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.501049951872035
For values of alpha =  0.0001 The log loss is: 0.6585278256347589
For values of alpha =  0.001 The log loss is: 0.5426775854305669
For values of alpha =  0.01 The log loss is: 0.4843166799186014
For values of alpha =  0.1 The log loss is: 0.6004019081854747
```

```
For values of alpha =  1 The log loss is: 0.6447654975374537
For values of alpha =  10 The log loss is: 0.6585278256347589
```




```
For values of best alpha =  0.01 The train log loss is: 0.4829428560579652
For values of best alpha =  0.01 The test log loss is: 0.4843166799186014
Total number of data points : 121287
```



4.6 XGBoost

In [22]:

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

X_train_w2v=X_train_w2v.as_matrix()
X_test_w2v=X_test_w2v.as_matrix()
y_train=y_train.as_matrix()

d_train = xgb.DMatrix(X_train_w2v, label=y_train)
d_test = xgb.DMatrix(X_test_w2v, label=y_test)

watchlist = [(d_train, 'train'), (d_test, 'valid')]

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, verbose_eval=10)

xgdmat = xgb.DMatrix(X_train_w2v,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

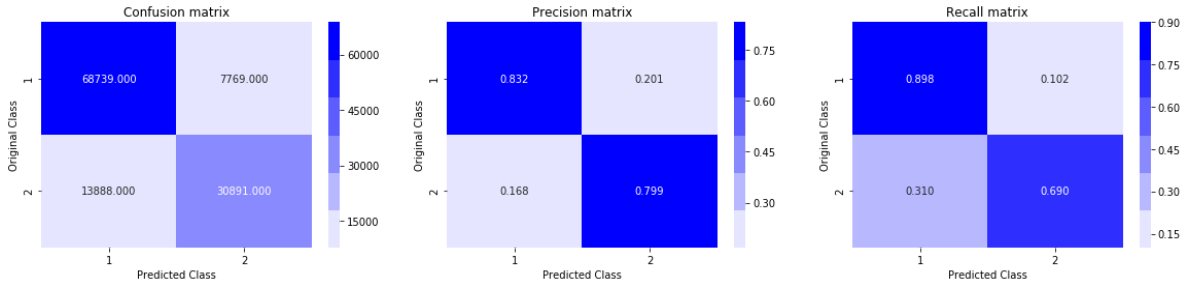
```
[0]     train-logloss:0.684853  valid-logloss:0.684882
Multiple eval metrics have been passed: 'valid-logloss' will be used for ear
ly stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.61512   valid-logloss:0.615607
[20]    train-logloss:0.563673  valid-logloss:0.56465
[30]    train-logloss:0.525481  valid-logloss:0.526675
[40]    train-logloss:0.496     valid-logloss:0.497506
[50]    train-logloss:0.473293  valid-logloss:0.474938
[60]    train-logloss:0.455034  valid-logloss:0.456835
[70]    train-logloss:0.440303  valid-logloss:0.442201
[80]    train-logloss:0.428369  valid-logloss:0.430382
[90]    train-logloss:0.418752  valid-logloss:0.4208
[100]   train-logloss:0.410769  valid-logloss:0.412868
[110]   train-logloss:0.404189  valid-logloss:0.406332
[120]   train-logloss:0.398658  valid-logloss:0.400871
[130]   train-logloss:0.393979  valid-logloss:0.396199
[140]   train-logloss:0.390023  valid-logloss:0.392263
[150]   train-logloss:0.386659  valid-logloss:0.388961
[160]   train-logloss:0.383575  valid-logloss:0.385933
[170]   train-logloss:0.381021  valid-logloss:0.383423
[180]   train-logloss:0.378699  valid-logloss:0.381139
[190]   train-logloss:0.376694  valid-logloss:0.379168
[200]   train-logloss:0.374709  valid-logloss:0.377248
[210]   train-logloss:0.373115  valid-logloss:0.37573
[220]   train-logloss:0.371503  valid-logloss:0.374186
[230]   train-logloss:0.369969  valid-logloss:0.372723
[240]   train-logloss:0.368654  valid-logloss:0.371469
[250]   train-logloss:0.367308  valid-logloss:0.370215
[260]   train-logloss:0.366084  valid-logloss:0.369091
[270]   train-logloss:0.364898  valid-logloss:0.367992
[280]   train-logloss:0.36367   valid-logloss:0.366893
[290]   train-logloss:0.362507  valid-logloss:0.365818
[300]   train-logloss:0.361337  valid-logloss:0.364775
[310]   train-logloss:0.360303  valid-logloss:0.36383
[320]   train-logloss:0.35934   valid-logloss:0.362955
```

```
[330]   train-logloss:0.358404  valid-logloss:0.362118
[340]   train-logloss:0.357523  valid-logloss:0.361353
[350]   train-logloss:0.356652  valid-logloss:0.360571
[360]   train-logloss:0.355855  valid-logloss:0.359882
[370]   train-logloss:0.355055  valid-logloss:0.359197
[380]   train-logloss:0.354295  valid-logloss:0.35852
[390]   train-logloss:0.353534  valid-logloss:0.357859
[399]   train-logloss:0.352891  valid-logloss:0.3573
The test log loss is: 0.3573001481963153
```

In [23]:

```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
Total number of data points : 121287
```



5. Assignments

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

In [3]:

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n
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	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	

In [4]:

Out[4]:

	id	qid1_x	qid2_x	question1_x	question2_x	is_duplicate_x	cwc_min	cwc_max	csc_min
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
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

2 rows × 37 columns

In [5]:

```
Y_tfidf = df_total["is_duplicate_x"]
X_tfidf= df_total.drop("is_duplicate_x", axis = 1)
```

In [6]:

```
train_x_tfidf, test_x_tfidf , train_y, test_y = train_test_split(X_tfidf, Y_tfidf, test_siz
print("Number of data points in train data :", train_x_tfidf.shape)
print("Number of data points in test data :", test_x_tfidf.shape)
```

```
Number of data points in train data : (283003, 36)
Number of data points in test data : (121287, 36)
```

In [7]:

```
train_x_tfidf.columns
```

Out[7]:

```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_word_e
q',
       'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio',
       'token_sort_ratio', 'fuzz_ratio', 'fuzz_partial_ratio',
       'longest_substr_ratio', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', 'freq_qid1', 'freq_qid2', 'q1len',
       'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Total',
       'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

train Features

In [78]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = np.array(train_x_tfidf['question1_x'].astype("unicode")) + np.array(train_x_tfi

tfidf = TfidfVectorizer(lowercase=False,ngram_range=(1,3),min_df=10)
tfidf.fit(questions)
tfidf_train =tfidf.transform(questions)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [79]:

```
len(tfidf.get_feature_names())
```

Out[79]:

```
122765
```

In [80]:

```
tfidf_train.shape
```

Out[80]:

```
(283003, 122765)
```

test features

In [81]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = np.array(test_x_tfidf['question1_x'].astype("unicode")) + np.array(test_x_tfidf

#tfidf = TfidfVectorizer(lowercase=False)
tfidf_test=tfidf.transform(questions)

# dict key:word and value:tf-idf score
word2tfidf_test = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [82]:

```
len(tfidf.get_feature_names())
```

Out[82]:

```
122765
```

In [83]:

```
tfidf_test.shape
```

Out[83]:

```
(121287, 122765)
```

In [84]:

```
train_feats = train_x_tfidf.drop(["id", "is_duplicate_y", "qid1_x", "qid2_x", "qid1_y", "qi
train_feats.shape
```

Out[84]:

```
(283003, 26)
```

In [85]:

```
test_feats = test_x_tfidf.drop(["id", "is_duplicate_y", "qid1_x", "qid2_x", "qid1_y", "qid2
test_feats.shape
```

Out[85]:

```
(121287, 26)
```

making the model

In [86]:

```
from scipy.sparse import hstack

X_train = hstack([train_feats,tfidf_train],format = "csr")
```

In [87]:

```
X_train.shape
```

Out[87]:

```
(283003, 122791)
```

In [88]:

```
train_y.shape
```

Out[88]:

```
(283003,)
```

In [89]:

```
X_test = hstack([test_feats,tfidf_test],format = "csr")
```

In [90]:

```
X_test.shape
```

Out[90]:

```
(121287, 122791)
```

In [91]:

```
test_y.shape
```

Out[91]:

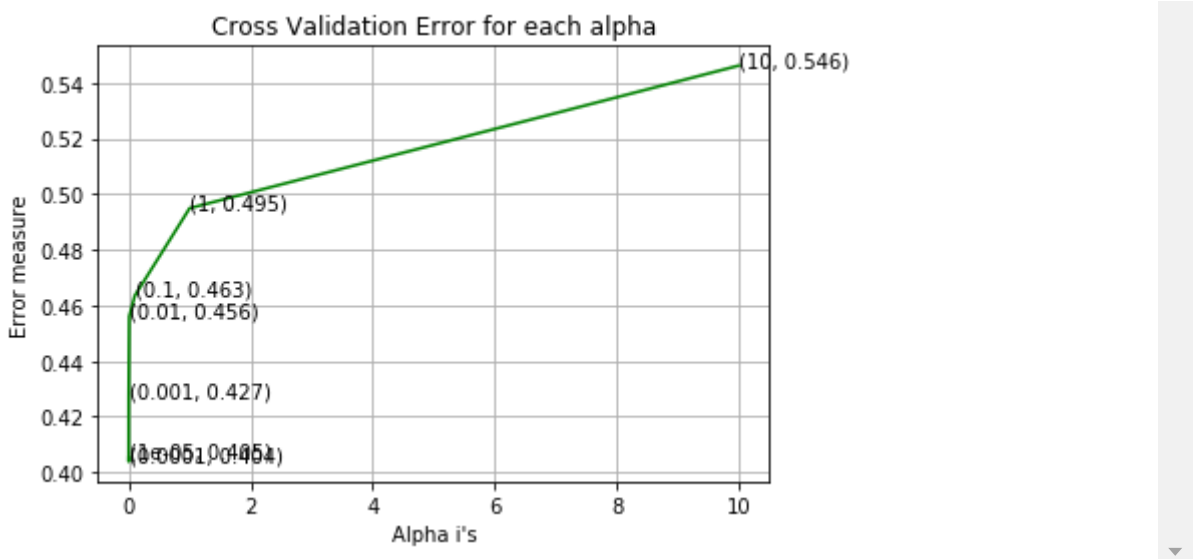
```
(121287,)
```

Logistic regression with tfidf vec

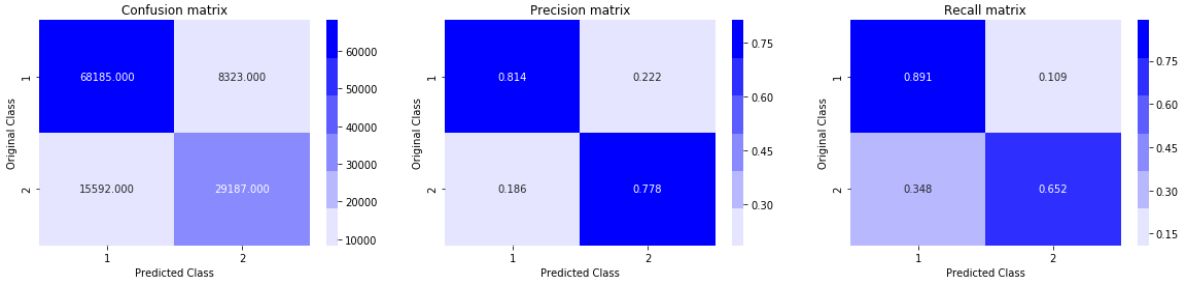
In [95]:

```
alpha = [10 ** x for x in range(-5, 2)] # 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_tfidf,train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_test, test_y)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(test_y, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(test_y, predict_y, label

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, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,train_y)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(t
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(te
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(test_y, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.4051610001597163
For values of alpha =  0.0001 The log loss is: 0.4038326200076756
For values of alpha =  0.001 The log loss is: 0.4267966850661711
For values of alpha =  0.01 The log loss is: 0.4559996558368846
For values of alpha =  0.1 The log loss is: 0.4634252450815399
For values of alpha =  1 The log loss is: 0.49490282801765517
For values of alpha =  10 The log loss is: 0.5461463590609442
```




```
For values of best alpha =  0.0001 The train log loss is: 0.3911013998478218
For values of best alpha =  0.0001 The test log loss is: 0.3940003195746835
Total number of data points : 121287
```



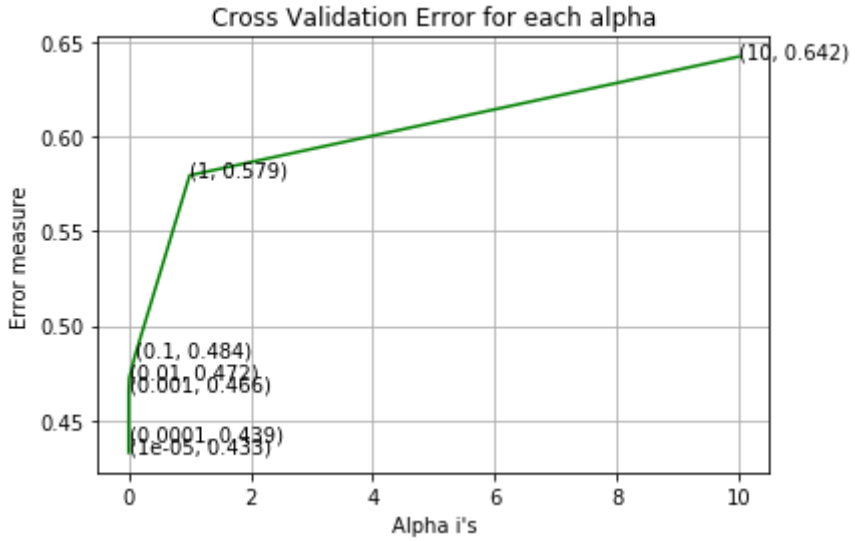
linear SVM with tfidf

In [96]:

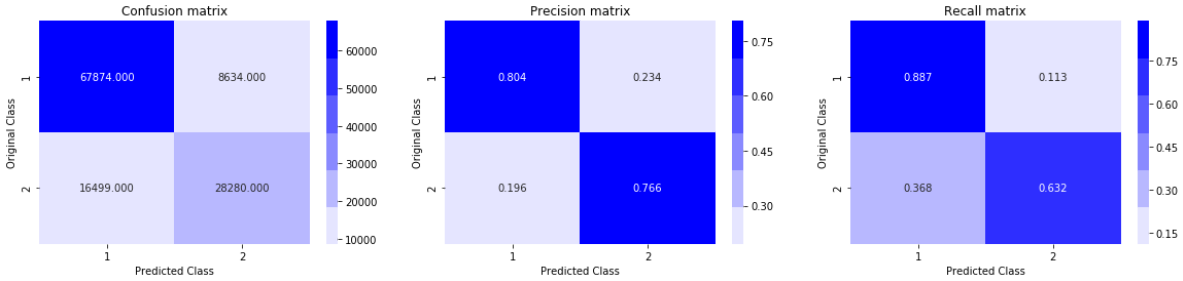
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Desce
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link:
#-------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train,train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,train_y)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(test_y, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(test_y, predict_y, label

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='l1', loss='hinge', random_state=42)
clf.fit(X_train_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, train_y)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(t
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(te
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(test_y, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.43282936463390015
For values of alpha =  0.0001 The log loss is: 0.43899659592973195
For values of alpha =  0.001 The log loss is: 0.4656200932810228
For values of alpha =  0.01 The log loss is: 0.4722079231878463
```

```
For values of alpha =  0.1 The log loss is: 0.48369348793307015
For values of alpha =  1 The log loss is: 0.5794922263519543
For values of alpha =  10 The log loss is: 0.6422913108070278
```




```
For values of best alpha =  1e-05 The train log loss is: 0.43003130493162606
For values of best alpha =  1e-05 The test log loss is: 0.43282936463390015
Total number of data points : 121287
```



Hyperparameter tunning using RandomSearch

In [19]:

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
from tqdm import tqdm

param_grid = {"max_depth":[1,5,10,50,100,500,1000],
              "n_estimators":[50,100,150,200,300,400,500]}

xgb_model = RandomizedSearchCV(xgb.XGBClassifier(n_jobs=-1,random_state=25), param_distribu
xgb_model.fit(X_train,y_train)
xgb_model.best_params_
```

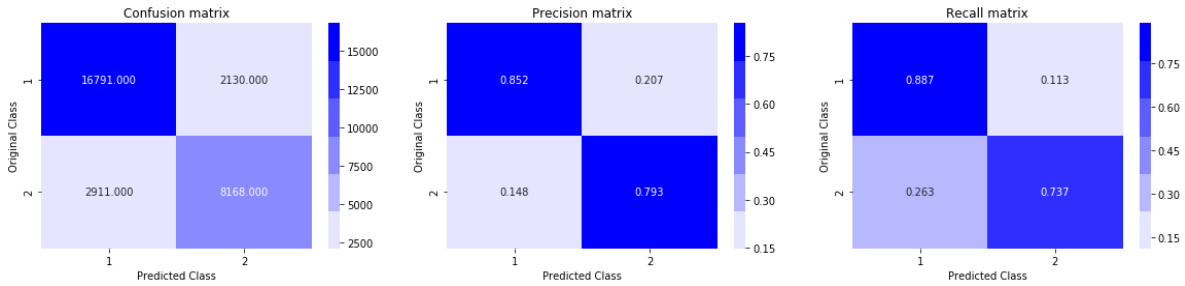
Out[19]:

```
{'max_depth': 5, 'n_estimators': 400}
```

In [21]:

```
clf=xgb.XGBClassifier(n_jobs=-1,random_state=25,max_depth=5,n_estimators=400)
clf.fit(X_train,y_train)
y_pred_test=clf.predict_proba(X_test)
y_pred_train=clf.predict_proba(X_train)
log_loss_train = log_loss(y_train, y_pred_train, eps=1e-15)
log_loss_test=log_loss(y_test,y_pred_test,eps=1e-15)
print('Train log loss = ',log_loss_train,' Test log loss = ',log_loss_test)
predicted_y=np.argmax(y_pred_test,axis=1)
plot_confusion_matrix(y_test,predicted_y)
```

```
Train log loss =  0.2295316629876758  Test log loss =  0.3351007574382394
```



In [24]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Feature Vectors","log loss"]
x.add_row(['Logistic regression','TFIDF w2v','0.437'])
x.add_row(['Linear SVM','TFIDF w2v','0.484'])
x.add_row(['XGBOOST','TFIDF w2v','0.357'])
x.add_row(['Logistic regression','TFIDF ','0.394'])
x.add_row(['Linear SVM','TFIDF','0.432'])
x.add_row(['XGBOOST','TFIDF w2v ','0.3351'])

print(x)
```

```
+---------------------+-----------------+----------+
|        Model        | Feature Vectors | log loss |
+---------------------+-----------------+----------+
| Logistic regression |    TFIDF w2v    |  0.437   |
|      Linear SVM     |    TFIDF w2v    |  0.484   |
|       XGBOOST       |    TFIDF w2v    |  0.357   |
| Logistic regression |      TFIDF      |  0.394   |
|      Linear SVM     |      TFIDF      |  0.432   |
|       XGBOOST       |    TFIDF w2v    |  0.3351  |
+---------------------+-----------------+----------+
```

In [ ]: