In [3]:

```
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.model_selection 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
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [4]:

```
#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','cwc_min','
cwc_max','csc_min','csc_max','ctc_min','ctc_max','last_word_eq','first_word_eq','abs_len_diff','me
an_len','token_set_ratio','token_sort_ratio','fuzz_ratio','fuzz_partial_ratio','longest_substr_rati
o','freq_qid1','freq_qid2','q1len','q2len','q1_n_words','q2_n_words','word_Common','word_Total','w
ord_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','16_x','17_x','18_x','19_x','20_x','21_x','22_x','23_x','24_x','25_x','26_x','27_x','28_x','
29_x','30_x','31_x','32_x','33_x','34_x','35_x','36_x','37_x','38_x','39_x','40_x','41_x','42_x','4
```

```
3_x','44_x','45_x','46_x','47_x','48_x','49_x','50_x','51_x','52_x','53_x','54_x','55_x','56_x','57
_x','58_x','59_x','60_x','61_x','62_x','63_x','64_x','65_x','66_x','67_x','68_x','69_x','70_x','71_
x','72_x','73_x','74_x','75_x','76_x','77_x','78_x','79_x','80_x','81_x','82_x','83_x','84_x','85_x
','86_x','87_x','88_x','89_x','90_x','91_x','92_x','93_x','94_x','95_x','96_x','97_x','98_x','99_x'
,'100_x','101_x','102_x','103_x','104_x','105_x','106_x','107_x','108_x','109_x','110_x','111_x','
112_x','113_x','114_x','115_x','116_x','117_x','118_x','119_x','120_x','121_x','122_x','123_x','12
4_x','125_x','126_x','127_x','128_x','129_x','130_x','131_x','132_x','133_x','134_x','135_x','136_
x','137_x','138_x','139_x','140_x','141_x','142_x','143_x','144_x','145_x','146_x','147_x','148_x'
,'149_x','150_x','151_x','152_x','153_x','154_x','155_x','156_x','157_x','158_x','159_x','160_x','
161_x','162_x','163_x','164_x','165_x','166_x','167_x','168_x','169_x','170_x','171_x','172_x','17
3_x','174_x','175_x','176_x','177_x','178_x','179_x','180_x','181_x','182_x','183_x','184_x','185_
x','186_x','187_x','188_x','189_x','190_x','191_x','192_x','193_x','194_x','195_x','196_x','197_x'
,'198_x','199_x','200_x','201_x','202_x','203_x','204_x','205_x','206_x','207_x','208_x','209_x','
210_x','211_x','212_x','213_x','214_x','215_x','216_x','217_x','218_x','219_x','220_x','221_x','22
2_x','223_x','224_x','225_x','226_x','227_x','228_x','229_x','230_x','231_x','232_x','233_x','234_
x','235_x','236_x','237_x','238_x','239_x','240_x','241_x','242_x','243_x','244_x','245_x','246_x'
,'247_x','248_x','249_x','250_x','251_x','252_x','253_x','254_x','255_x','256_x','257_x','258_x','
259_x','260_x','261_x','262_x','263_x','264_x','265_x','266_x','267_x','268_x','269_x','270_x','27
1_x','272_x','273_x','274_x','275_x','276_x','277_x','278_x','279_x','280_x','281_x','282_x','283_
x','284_x','285_x','286_x','287_x','288_x','289_x','290_x','291_x','292_x','293_x','294_x','295_x'
,'296_x','297_x','298_x','299_x','300_x','301_x','302_x','303_x','304_x','305_x','306_x','307_x','
308_x','309_x','310_x','311_x','312_x','313_x','314_x','315_x','316_x','317_x','318_x','319_x','32
0_x','321_x','322_x','323_x','324_x','325_x','326_x','327_x','328_x','329_x','330_x','331_x','332_
x','333_x','334_x','335_x','336_x','337_x','338_x','339_x','340_x','341_x','342_x','343_x','344_x'
,'345_x','346_x','347_x','348_x','349_x','350_x','351_x','352_x','353_x','354_x','355_x','356_x','
357_x','358_x','359_x','360_x','361_x','362_x','363_x','364_x','365_x','366_x','367_x','368_x','36
9_x','370_x','371_x','372_x','373_x','374_x','375_x','376_x','377_x','378_x','379_x','380_x','381_
x','382_x','383_x','0_y','1_y','2_y','3_y','4_y','5_y','6_y','7_y','8_y','9_y','10_y','11_y','12_y'
,'13_y','14_y','15_y','16_y','17_y','18_y','19_y','20_y','21_y','22_y','23_y','24_y','25_y','26_y',
'27_y','28_y','29_y','30_y','31_y','32_y','33_y','34_y','35_y','36_y','37_y','38_y','39_y','40_y','
41_y','42_y','43_y','44_y','45_y','46_y','47_y','48_y','49_y','50_y','51_y','52_y','53_y','54_y','5
5_y','56_y','57_y','58_y','59_y','60_y','61_y','62_y','63_y','64_y','65_y','66_y','67_y','68_y','69
_y','70_y','71_y','72_y','73_y','74_y','75_y','76_y','77_y','78_y','79_y','80_y','81_y','82_y','83_
y','84_y','85_y','86_y','87_y','88_y','89_y','90_y','91_y','92_y','93_y','94_y','95_y','96_y','97_y
','98_y','99_y','100_y','101_y','102_y','103_y','104_y','105_y','106_y','107_y','108_y','109_y','11
0_y','111_y','112_y','113_y','114_y','115_y','116_y','117_y','118_y','119_y','120_y','121_y','122_
y','123_y','124_y','125_y','126_y','127_y','128_y','129_y','130_y','131_y','132_y','133_y','134_y'
,'135_y','136_y','137_y','138_y','139_y','140_y','141_y','142_y','143_y','144_y','145_y','146_y','
147_y','148_y','149_y','150_y','151_y','152_y','153_y','154_y','155_y','156_y','157_y','158_y','15
9_y','160_y','161_y','162_y','163_y','164_y','165_y','166_y','167_y','168_y','169_y','170_y','171_
y','172_y','173_y','174_y','175_y','176_y','177_y','178_y','179_y','180_y','181_y','182_y','183_y'
,'184_y','185_y','186_y','187_y','188_y','189_y','190_y','191_y','192_y','193_y','194_y','195_y','
196_y','197_y','198_y','199_y','200_y','201_y','202_y','203_y','204_y','205_y','206_y','207_y','20
8_y','209_y','210_y','211_y','212_y','213_y','214_y','215_y','216_y','217_y','218_y','219_y','220_
y','221_y','222_y','223_y','224_y','225_y','226_y','227_y','228_y','229_y','230_y','231_y','232_y'
,'233_y','234_y','235_y','236_y','237_y','238_y','239_y','240_y','241_y','242_y','243_y','244_y','
245_y','246_y','247_y','248_y','249_y','250_y','251_y','252_y','253_y','254_y','255_y','256_y','25
7_y','258_y','259_y','260_y','261_y','262_y','263_y','264_y','265_y','266_y','267_y','268_y','269_
y','270_y','271_y','272_y','273_y','274_y','275_y','276_y','277_y','278_y','279_y','280_y','281_y'
,'282_y','283_y','284_y','285_y','286_y','287_y','288_y','289_y','290_y','291_y','292_y','293_y','
294_y','295_y','296_y','297_y','298_y','299_y','300_y','301_y','302_y','303_y','304_y','305_y','30
6_y','307_y','308_y','309_y','310_y','311_y','312_y','313_y','314_y','315_y','316_y','317_y','318_
y','319_y','320_y','321_y','322_y','323_y','324_y','325_y','326_y','327_y','328_y','329_y','330_y'
,'331_y','332_y','333_y','334_y','335_y','336_y','337_y','338_y','339_y','340_y','341_y','342_y','
343_y','344_y','345_y','346_y','347_y','348_y','349_y','350_y','351_y','352_y','353_y','354_y','35
5_y','356_y','357_y','358_y','359_y','360_y','361_y','362_y','363_y','364_y','365_y','366_y','367_
y','368_y','369_y','370_y','371_y','372_y','373_y','374_y','375_y','376_y','377_y','378_y','379_y'
,'380_y','381_y','382_y','383_y'], chunksize=chunksize, iterator=True, encoding='utf-8', ):
        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 [5]:

```
#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
```

```
        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 [6]:

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

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

In [7]:

```
# 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_r)
        conn_r.commit()
        conn_r.close()
```

In [8]:

```
# 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 [9]:

```
data.head()
```

Out[9]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
1	0.199996000079998	0.166663888935184	0.66664444518516	0.333327777870369	0.374995312558593	0.249997916684028	0.0
2	0.749981250468738	0.749981250468738	0.499975001249937	0.499975001249937	0.666655555740738	0.666655555740738	1.0
3	0.599988000239995	0.249997916684028	0.249993750156246	0.111109876556927	0.444439506227709	0.173912287337881	0.0
4	0.333327777870369	0.285710204139941	0.0	0.0	0.166665277789352	0.142856122456268	0.0
5	0.666655555740738	0.666655555740738	0.499987500312492	0.249996875039062	0.599994000059999	0.428568367368804	0.0

5 rows × 794 columns

4.2 Converting strings to numerics

4.2 Converting strings to numerics

In [10]:

```
# 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)
```

```
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
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
16_x
17_x
18_x
19_x
20_x
21_x
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
```

```
39_x
40_x
41_x
42_x
43_x
44_x
45_x
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
65_x
66_x
67_x
68_x
69_x
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
96_x
97_x
98_x
99_x
100_x
101_x
102_x
103_x
104_x
105_x
106_x
107_x
108_x
109_x
110_x
111_x
112_x
113_x
114_x
115_x
116_x
```

```
116_x
117_x
118_x
119_x
120_x
121_x
122_x
123_x
124_x
125_x
126_x
127_x
128_x
129_x
130_x
131_x
132_x
133_x
134_x
135_x
136_x
137_x
138_x
139_x
140_x
141_x
142_x
143_x
144_x
145_x
146_x
147_x
148_x
149_x
150_x
151_x
152_x
153_x
154_x
155_x
156_x
157_x
158_x
159_x
160_x
161_x
162_x
163_x
164_x
165_x
166_x
167_x
168_x
169_x
170_x
171_x
172_x
173_x
174_x
175_x
176_x
177_x
178_x
179_x
180_x
181_x
182_x
183_x
184_x
185_x
186_x
187_x
188_x
189_x
190_x
191_x
192_x
```

193_x
194_x
195_x
196_x
197_x
198_x
199_x
200_x
201_x
202_x
203_x
204_x
205_x
206_x
207_x
208_x
209_x
210_x
211_x
212_x
213_x
214_x
215_x
216_x
217_x
218_x
219_x
220_x
221_x
222_x
223_x
224_x
225_x
226_x
227_x
228_x
229_x
230_x
231_x
232_x
233_x
234_x
235_x
236_x
237_x
238_x
239_x
240_x
241_x
242_x
243_x
244_x
245_x
246_x
247_x
248_x
249_x
250_x
251_x
252_x
253_x
254_x
255_x
256_x
257_x
258_x
259_x
260_x
261_x
262_x
263_x
264_x
265_x
266_x
267_x
268_x
269_x

```
270_x
271_x
272_x
273_x
274_x
275_x
276_x
277_x
278_x
279_x
280_x
281_x
282_x
283_x
284_x
285_x
286_x
287_x
288_x
289_x
290_x
291_x
292_x
293_x
294_x
295_x
296_x
297_x
298_x
299_x
300_x
301_x
302_x
303_x
304_x
305_x
306_x
307_x
308_x
309_x
310_x
311_x
312_x
313_x
314_x
315_x
316_x
317_x
318_x
319_x
320_x
321_x
322_x
323_x
324_x
325_x
326_x
327_x
328_x
329_x
330_x
331_x
332_x
333_x
334_x
335_x
336_x
337_x
338_x
339_x
340_x
341_x
342_x
343_x
344_x
345_x
346_x
```

```
347_x
348_x
349_x
350_x
351_x
352_x
353_x
354_x
355_x
356_x
357_x
358_x
359_x
360_x
361_x
362_x
363_x
364_x
365_x
366_x
367_x
368_x
369_x
370_x
371_x
372_x
373_x
374_x
375_x
376_x
377_x
378_x
379_x
380_x
381_x
382_x
383_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
22_y
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
```

```
40_y
41_y
42_y
43_y
44_y
45_y
46_y
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
96_y
97_y
98_y
99_y
100_y
101_y
102_y
103_y
104_y
105_y
106_y
107_y
108_y
109_y
110_y
111_y
112_y
113_y
114_y
115_y
116_y
```

```
117_y
118_y
119_y
120_y
121_y
122_y
123_y
124_y
125_y
126_y
127_y
128_y
129_y
130_y
131_y
132_y
133_y
134_y
135_y
136_y
137_y
138_y
139_y
140_y
141_y
142_y
143_y
144_y
145_y
146_y
147_y
148_y
149_y
150_y
151_y
152_y
153_y
154_y
155_y
156_y
157_y
158_y
159_y
160_y
161_y
162_y
163_y
164_y
165_y
166_y
167_y
168_y
169_y
170_y
171_y
172_y
173_y
174_y
175_y
176_y
177_y
178_y
179_y
180_y
181_y
182_y
183_y
184_y
185_y
186_y
187_y
188_y
189_y
190_y
191_y
192_y
193_y
```

```
194_y
195_y
196_y
197_y
198_y
199_y
200_y
201_y
202_y
203_y
204_y
205_y
206_y
207_y
208_y
209_y
210_y
211_y
212_y
213_y
214_y
215_y
216_y
217_y
218_y
219_y
220_y
221_y
222_y
223_y
224_y
225_y
226_y
227_y
228_y
229_y
230_y
231_y
232_y
233_y
234_y
235_y
236_y
237_y
238_y
239_y
240_y
241_y
242_y
243_y
244_y
245_y
246_y
247_y
248_y
249_y
250_y
251_y
252_y
253_y
254_y
255_y
256_y
257_y
258_y
259_y
260_y
261_y
262_y
263_y
264_y
265_y
266_y
267_y
268_y
269_y
270_y
```

```
271_y
272_y
273_y
274_y
275_y
276_y
277_y
278_y
279_y
280_y
281_y
282_y
283_y
284_y
285_y
286_y
287_y
288_y
289_y
290_y
291_y
292_y
293_y
294_y
295_y
296_y
297_y
298_y
299_y
300_y
301_y
302_y
303_y
304_y
305_y
306_y
307_y
308_y
309_y
310_y
311_y
312_y
313_y
314_y
315_y
316_y
317_y
318_y
319_y
320_y
321_y
322_y
323_y
324_y
325_y
326_y
327_y
328_y
329_y
330_y
331_y
332_y
333_y
334_y
335_y
336_y
337_y
338_y
339_y
340_y
341_y
342_y
343_y
344_y
345_y
346_y
347_y
```

```
348_y
349_y
350_y
351_y
352_y
353_y
354_y
355_y
356_y
357_y
358_y
359_y
360_y
361_y
362_y
363_y
364_y
365_y
366_y
367_y
368_y
369_y
370_y
371_y
372_y
373_y
374_y
375_y
376_y
377_y
378_y
379_y
380_y
381_y
382_y
383_y
```

In [11]:

```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
y_true = list(map(int, y_true.values))
```

4.3 Random train test split(70:30)

In [12]:

```
X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

In [13]:

```
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 : (70000, 794)
Number of data points in test data : (30000, 794)
```

In [14]:

```
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 train 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:  0.6319285714285714 Class 1:  0.36807142857142855
---------- Distribution of output variable in train data ----------
```

```
Class 0:  0.36806666666666665 Class 1:  0.36806666666666665
```

In [15]:

```python
# 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 = [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()
```

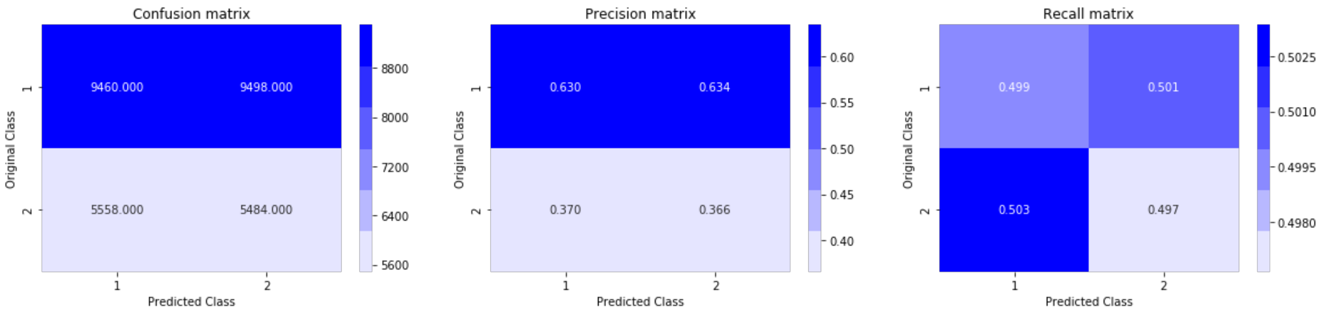
4.4 Building a random model (Finding worst-case log-loss)

In [16]:

```python
# 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 0.8920980072030672
```



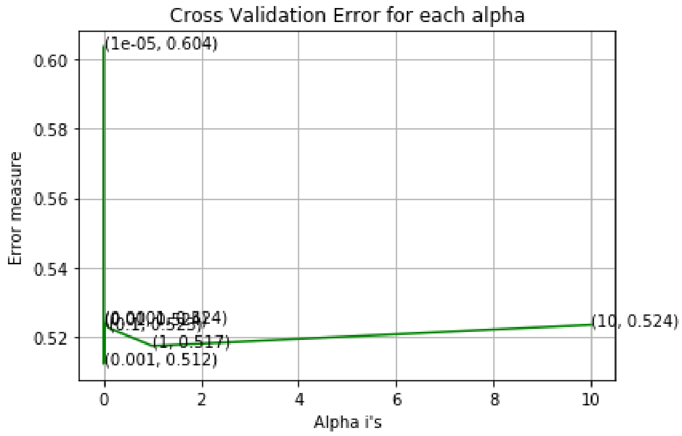
4.4 Logistic Regression with hyperparameter tuning

In [17]:

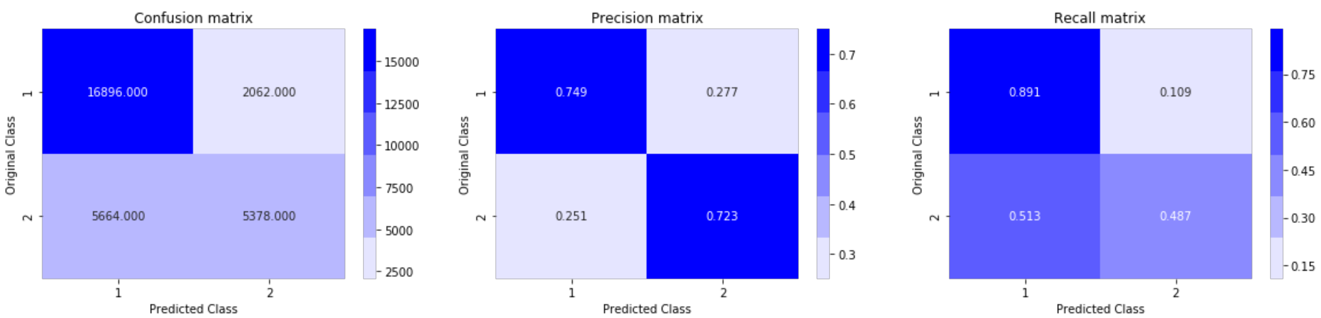
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# 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)
    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.cl
asses_, 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)
```

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

```
For values of alpha =  1e-05 The log loss is: 0.6035913054302467
For values of alpha =  0.0001 The log loss is: 0.5243625576325577
For values of alpha =  0.001 The log loss is: 0.5122626091429161
For values of alpha =  0.01 The log loss is: 0.5241346914793784
For values of alpha =  0.1 The log loss is: 0.5226398430372254
For values of alpha =  1 The log loss is: 0.5174211365837058
For values of alpha =  10 The log loss is: 0.5235052874500619
```




```
For values of best alpha =  0.001 The train log loss is: 0.505494765793637
For values of best alpha =  0.001 The test log loss is: 0.5122626091429161
Total number of data points : 30000
```



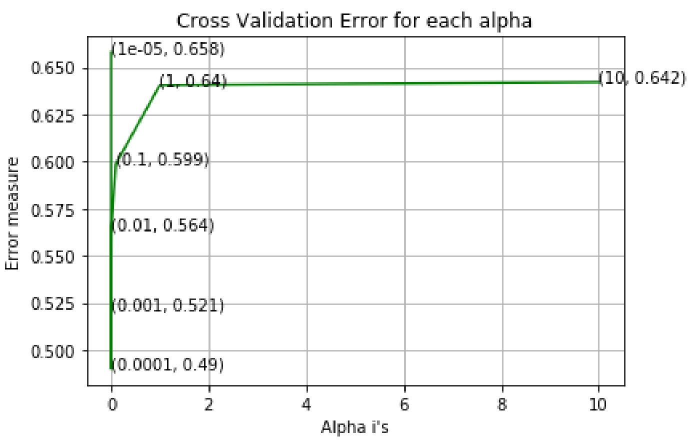
4.5 Linear SVM with hyperparameter tuning

In [18]:

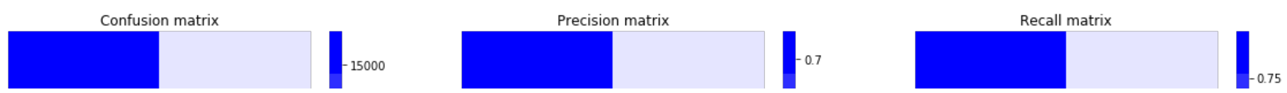
```python
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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 Descent.
# predict(X) Predict class labels for samples in X.

#-------------------------------
```

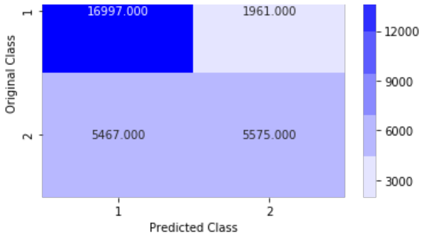
```
# video link:
#------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, 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.cl
asses_, 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='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_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_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)
```

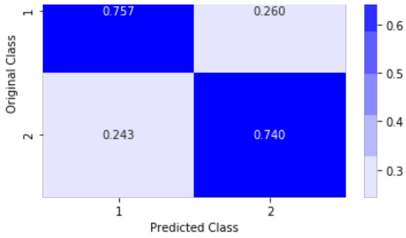
```
For values of alpha =  1e-05 The log loss is: 0.6579187053993628
For values of alpha =  0.0001 The log loss is: 0.489938923497778
For values of alpha =  0.001 The log loss is: 0.5214825234723068
For values of alpha =  0.01 The log loss is: 0.5642511907598364
For values of alpha =  0.1 The log loss is: 0.598593674111375
For values of alpha =  1 The log loss is: 0.640443364230696
For values of alpha =  10 The log loss is: 0.6420695451380629
```

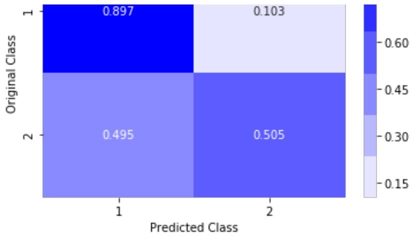



```
For values of best alpha =  0.0001 The train log loss is: 0.4784507404577099
For values of best alpha =  0.0001 The test log loss is: 0.489938923497778
Total number of data points : 30000
```









4.6 XGBoost

In [19]:

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

d_train = xgb.DMatrix(X_train, label=y_train)
d_test = xgb.DMatrix(X_test, 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,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))
```

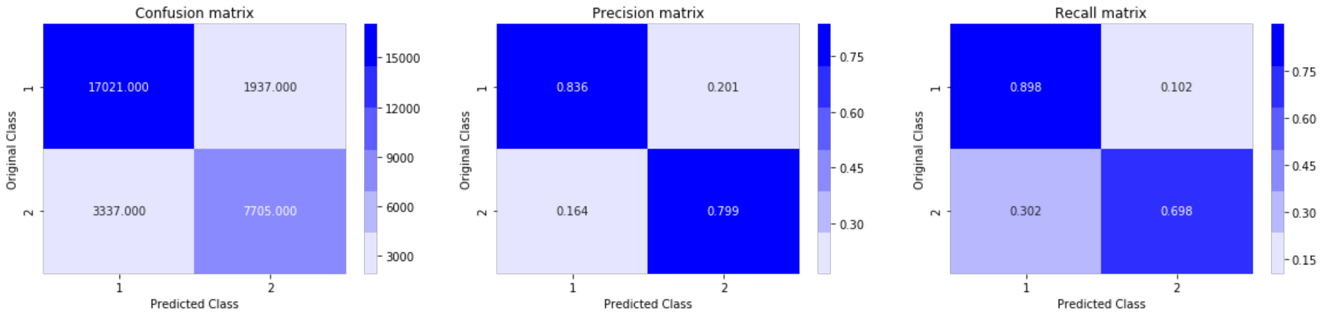
```
[0] train-logloss:0.68484 valid-logloss:0.684882
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10] train-logloss:0.616115 valid-logloss:0.616505
[20] train-logloss:0.564948 valid-logloss:0.565475
[30] train-logloss:0.526655 valid-logloss:0.527422
[40] train-logloss:0.496822 valid-logloss:0.497844
[50] train-logloss:0.473779 valid-logloss:0.475191
[60] train-logloss:0.455329 valid-logloss:0.456965
[70] train-logloss:0.44052 valid-logloss:0.442376
[80] train-logloss:0.428356 valid-logloss:0.43041
[90] train-logloss:0.418476 valid-logloss:0.420766
[100] train-logloss:0.410409 valid-logloss:0.412846
[110] train-logloss:0.403529 valid-logloss:0.406185
[120] train-logloss:0.397776 valid-logloss:0.400636
[130] train-logloss:0.392836 valid-logloss:0.395936
[140] train-logloss:0.388292 valid-logloss:0.391676
[150] train-logloss:0.384757 valid-logloss:0.388449
[160] train-logloss:0.38155 valid-logloss:0.385492
[170] train-logloss:0.37865 valid-logloss:0.382863
[180] train-logloss:0.376222 valid-logloss:0.38061
[190] train-logloss:0.373858 valid-logloss:0.378418
[200] train-logloss:0.371746 valid-logloss:0.376525
[210] train-logloss:0.369602 valid-logloss:0.374613
[220] train-logloss:0.367389 valid-logloss:0.372679
[230] train-logloss:0.365485 valid-logloss:0.371085
[240] train-logloss:0.363611 valid-logloss:0.369498
[250] train-logloss:0.361767 valid-logloss:0.367944
[260] train-logloss:0.360015 valid-logloss:0.366558
[270] train-logloss:0.358499 valid-logloss:0.365352
[280] train-logloss:0.357023 valid-logloss:0.364254
[290] train-logloss:0.35554 valid-logloss:0.363127
[300] train-logloss:0.354104 valid-logloss:0.362084
[310] train-logloss:0.352855 valid-logloss:0.361177
[320] train-logloss:0.351566 valid-logloss:0.360249
[330] train-logloss:0.350334 valid-logloss:0.359345
[340] train-logloss:0.349123 valid-logloss:0.358529
[350] train-logloss:0.347998 valid-logloss:0.35775
[360] train-logloss:0.346989 valid-logloss:0.357056
[370] train-logloss:0.345849 valid-logloss:0.356341
[380] train-logloss:0.344787 valid-logloss:0.355725
```

```
[390] train-logloss:0.343777 valid-logloss:0.355069
[399] train-logloss:0.342925 valid-logloss:0.354573
The test log loss is: 0.35457332921151535
```

In [20]:

```
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 : 30000
```



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 [21]:

```
df = pd.read_csv('train.csv')
```

In [22]:

```
df.head()
```

Out[22]:

	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 [23]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

In [51]:

```
nlpfeatures = pd.read_csv('nlp_features_train.csv', encoding = 'latin-1')
```

In [53]:

```
nlpfeatures.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 21 columns):
id                      404290 non-null int64
qid1                    404290 non-null int64
qid2                    404290 non-null int64
question1               404276 non-null object
question2               404284 non-null object
is_duplicate            404290 non-null int64
cwc_min                 404290 non-null float64
cwc_max                 404290 non-null float64
csc_min                 404290 non-null float64
csc_max                 404290 non-null float64
ctc_min                 404290 non-null float64
ctc_max                 404290 non-null float64
last_word_eq            404290 non-null float64
first_word_eq           404290 non-null float64
abs_len_diff            404290 non-null float64
mean_len                404290 non-null float64
token_set_ratio         404290 non-null int64
token_sort_ratio        404290 non-null int64
fuzz_ratio              404290 non-null int64
fuzz_partial_ratio      404290 non-null int64
longest_substr_ratio    404290 non-null float64
dtypes: float64(11), int64(8), object(2)
memory usage: 64.8+ MB
```

In [54]:

```
# Filling the null values with ' '
nlpfeatures = nlpfeatures.fillna('')
nan_rows = nlpfeatures[nlpfeatures.isnull().any(1)]
```

In [55]:

```
nlpfeatures.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 21 columns):
id                      404290 non-null int64
qid1                    404290 non-null int64
qid2                    404290 non-null int64
question1               404290 non-null object
question2               404290 non-null object
is_duplicate            404290 non-null int64
cwc_min                 404290 non-null float64
cwc_max                 404290 non-null float64
csc_min                 404290 non-null float64
csc_max                 404290 non-null float64
ctc_min                 404290 non-null float64
ctc_max                 404290 non-null float64
last_word_eq            404290 non-null float64
first_word_eq           404290 non-null float64
abs_len_diff            404290 non-null float64
mean_len                404290 non-null float64
token_set_ratio         404290 non-null int64
token_sort_ratio        404290 non-null int64
fuzz_ratio              404290 non-null int64
fuzz_partial_ratio      404290 non-null int64
longest_substr_ratio    404290 non-null float64
dtypes: float64(11), int64(8), object(2)
memory usage: 64.8+ MB
```

In [207]:

```
X = nlpfeatures.drop(['is_duplicate', 'id', 'qid1', 'qid2'], axis = 1 )
y = nlpfeatures['is_duplicate']
```

In [208]:

```
X.head()
```

Out[208]:

	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mea
0	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	
1	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	
2	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	
3	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	
4	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	

In [209]:

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

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, 17)
Number of data points in test data : (121287, 17)
```

In [210]:

```
vectorizer = TfidfVectorizer(min_df = 3)
X_train_question1_tfidf = vectorizer.fit_transform(X_train['question1'].values)
X_test_question1_tfidf = vectorizer.transform(X_test['question1'].values)
```

In [211]:

```
vectorizer = TfidfVectorizer(min_df = 3)
X_train_question2_tfidf = vectorizer.fit_transform(X_train['question2'].values)
X_test_question2_tfidf = vectorizer.transform(X_test['question2'].values)
```

In [212]:

```
X_train.columns
```

Out[212]:

```
Index(['question1', 'question2', 'cwc_min', 'cwc_max', 'csc_min', 'csc_max',
       'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq', 'abs_len_diff',
       'mean_len', 'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [213]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
cwc_min_train = normalizer.fit_transform(X_train['cwc_min'].values.reshape(-1,1))
cwc_min_test = normalizer.transform((X_test['cwc_min'].values.reshape(-1,1)))

normalizer = Normalizer()
cwc_max_train = normalizer.fit_transform(X_train['cwc_max'].values.reshape(-1,1))
cwc_max_test = normalizer.transform(X_test['cwc_max'].values.reshape(-1,1))

normalizer = Normalizer()
csc_min_train = normalizer.fit_transform(X_train['csc_min'].values.reshape(-1,1))
csc_min_test = normalizer.transform(X_test['csc_min'].values.reshape(-1,1))

normalizer = Normalizer()
csc_max_train = normalizer.fit_transform(X_train['csc_max'].values.reshape(-1,1))
csc_max_test = normalizer.transform(X_test['csc_max'].values.reshape(-1,1))

normalizer = Normalizer()
ctc_min_train = normalizer.fit_transform(X_train['ctc_min'].values.reshape(-1,1))
ctc_min_test = normalizer.transform(X_test['ctc_min'].values.reshape(-1,1))

normalizer = Normalizer()
ctc_max_train = normalizer.fit_transform(X_train['ctc_max'].values.reshape(-1,1))
ctc_max_test = normalizer.transform(X_test['ctc_max'].values.reshape(-1,1))

normalizer = Normalizer()
last_word_eq_train = normalizer.fit_transform(X_train['last_word_eq'].values.reshape(-1,1))
last_word_eq_test = normalizer.transform(X_test['last_word_eq'].values.reshape(-1,1))

normalizer = Normalizer()
first_word_eq_train = normalizer.fit_transform(X_train['first_word_eq'].values.reshape(-1,1))
first_word_eq_test = normalizer.transform(X_test['first_word_eq'].values.reshape(-1,1))

normalizer = Normalizer()
abs_len_diff_train = normalizer.fit_transform(X_train['abs_len_diff'].values.reshape(-1,1))
abs_len_diff_test = normalizer.transform(X_test['abs_len_diff'].values.reshape(-1,1))

normalizer = Normalizer()
mean_len_train = normalizer.fit_transform(X_train['mean_len'].values.reshape(-1,1))
mean_len_test = normalizer.transform(X_test['mean_len'].values.reshape(-1,1))

normalizer = Normalizer()
token_set_ratio_train = normalizer.fit_transform(X_train['token_set_ratio'].values.reshape(-1,1))
token_set_ratio_test = normalizer.transform(X_test['token_set_ratio'].values.reshape(-1,1))

normalizer = Normalizer()
token_sort_ratio_train = normalizer.fit_transform(X_train['token_sort_ratio'].values.reshape(-1,1))
token_sort_ratio_test = normalizer.transform(X_test['token_sort_ratio'].values.reshape(-1,1))

normalizer = Normalizer()
fuzz_ratio_train = normalizer.fit_transform(X_train['fuzz_ratio'].values.reshape(-1,1))
fuzz_ratio_test = normalizer.transform(X_test['fuzz_ratio'].values.reshape(-1,1))

normalizer = Normalizer()
fuzz_partial_ratio_train = normalizer.fit_transform(X_train['fuzz_partial_ratio'].values.reshape(-1
,1))
fuzz_partial_ratio_test = normalizer.transform(X_test['fuzz_partial_ratio'].values.reshape(-1,1))

normalizer = Normalizer()
longest_substr_ratio_train =
normalizer.fit_transform(X_train['longest_substr_ratio'].values.reshape(-1,1))
longest_substr_ratio_test = normalizer.transform(X_test['longest_substr_ratio'].values.reshape(-1,1
))
```

In [215]:

```
from scipy.sparse import hstack
```

```
X_train = hstack((X_train_question1_tfidf, X_train_question2_tfidf, X_train_normalized,
cwc_min_train, cwc_max_train, csc_min_train, csc_max_train, ctc_min_train, ctc_max_train, last_word
_eq_train, first_word_eq_train, abs_len_diff_train, mean_len_train, token_set_ratio_train, token_so
rt_ratio_train, fuzz_ratio_train, fuzz_partial_ratio_train, longest_substr_ratio_train)).tocsr()
X_test = hstack((X_test_question1_tfidf, X_test_question2_tfidf, X_test_normalized, cwc_min_test,
cwc_max_test, csc_min_test, csc_max_test, ctc_min_test, ctc_max_test, last_word_eq_test,
first_word_eq_test, abs_len_diff_test, mean_len_test, token_set_ratio_test, token_sort_ratio_test,
fuzz_ratio_test, fuzz_partial_ratio_test, longest_substr_ratio_test)).tocsr()
```

In [217]:

```
X_test
```

Out[217]:

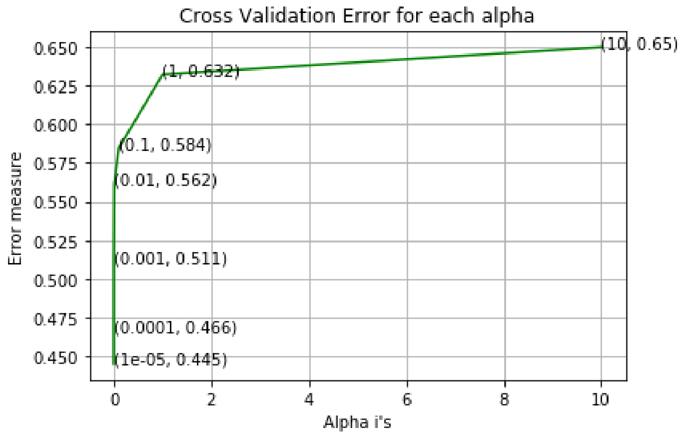
```
<121287x49186 sparse matrix of type '<class 'numpy.float64'>'
 with 4109274 stored elements in Compressed Sparse Row format>
```

In [218]:

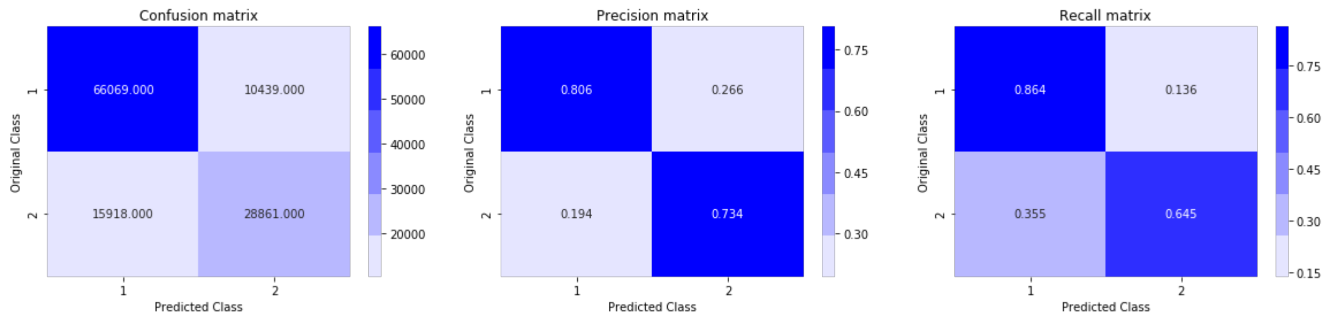
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# 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)
    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.cl
asses_, 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:",log_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, p
redict_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)
```

```
For values of alpha =  1e-05 The log loss is: 0.44526254212405547
For values of alpha =  0.0001 The log loss is: 0.4657760298871082
For values of alpha =  0.001 The log loss is: 0.5111258098893877
For values of alpha =  0.01 The log loss is: 0.5615102228944616
For values of alpha =  0.1 The log loss is: 0.5843290819582394
For values of alpha =  1 The log loss is: 0.6320509407231564
For values of alpha =  10 The log loss is: 0.6496734532520676
```




```
For values of best alpha =  1e-05 The train log loss is: 0.41816336872986865
For values of best alpha =  1e-05 The test log loss is: 0.44526254212405547
Total number of data points : 121287
```

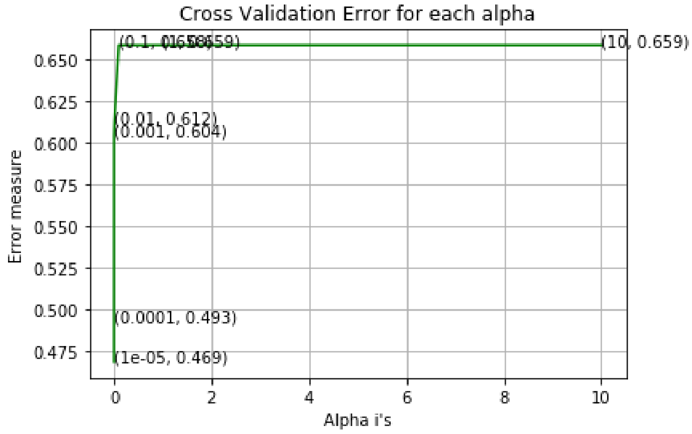


In [219]:

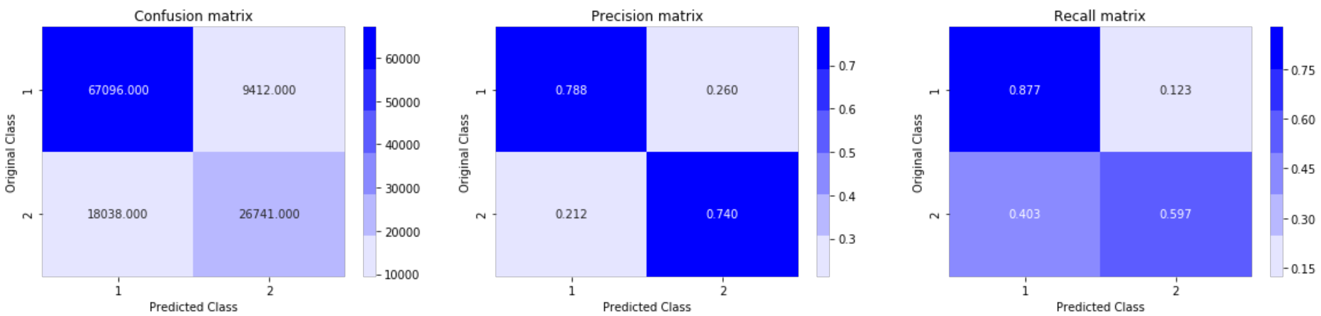
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# 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, 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.cl
asses_, 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='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_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_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)
```

```
For values of alpha =  1e-05 The log loss is: 0.4685881799916419
For values of alpha =  0.0001 The log loss is: 0.4925769044146379
For values of alpha =  0.001 The log loss is: 0.6044662447443142
For values of alpha =  0.01 The log loss is: 0.612134697211296
For values of alpha =  0.1 The log loss is: 0.6584974344080787
For values of alpha =  1 The log loss is: 0.658527825632271
For values of alpha =  10 The log loss is: 0.6585278256322541
```



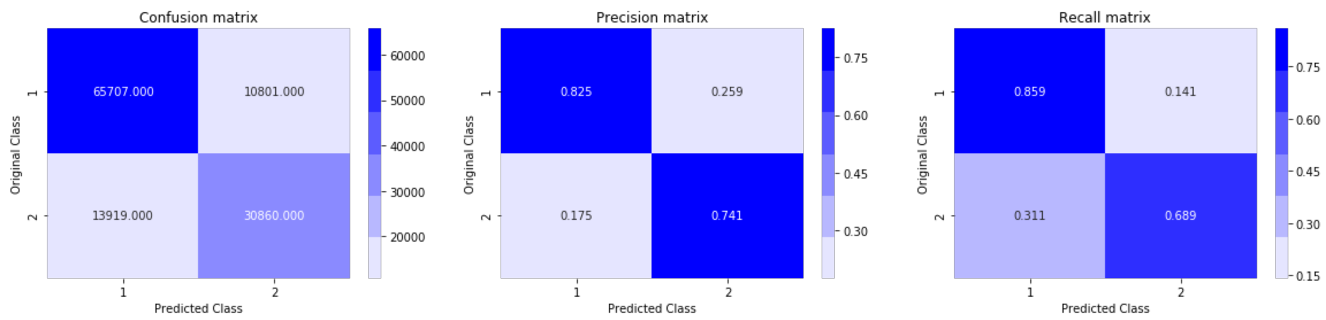

```
For values of best alpha =  1e-05 The train log loss is: 0.4494262203000246
For values of best alpha =  1e-05 The test log loss is: 0.4685881799916419
Total number of data points : 121287
```



In [238]:

```
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



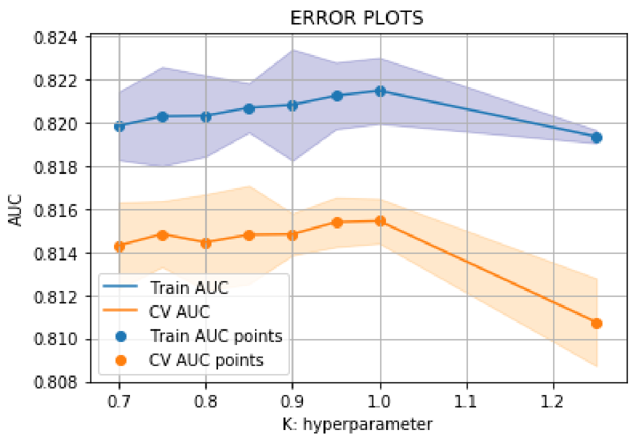
In [250]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://www.kaggle.com/phunter/xgboost-with-gridsearchcv
#The above link was looked at to implement xgboost with gridsearchcv
xgb_model = xgb.XGBClassifier()

#brute force scan for all parameters, here are the tricks
#usually max_depth is 6,7,8
#learning rate is around 0.05, but small changes may make big diff
#tuning min_child_weight subsample colsample_bytree can have
#much fun of fighting against overfit
#n_estimators is how many round of boosting
#finally, ensemble xgboost with multiple seeds may reduce variance
parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
              'objective':['binary:logistic'],
              'learning_rate': [0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1, 1.25], #so called `eta` value
              'max_depth': [6],
              'min_child_weight': [11],
              'silent': [1],
              'subsample': [0.8],
              'colsample_bytree': [0.7],
              'n_estimators': [10], #number of trees, change it to 1000 for better results
              'missing':[-999],
              'seed': [1337]}


clf = GridSearchCV(xgb_model, parameters, n_jobs=5,
                   cv=3,
                   scoring='roc_auc',
                   verbose=2, refit=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']

plt.plot(parameters['learning_rate'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['learning_rate'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['learning_rate'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['learning_rate'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')

plt.scatter(parameters['learning_rate'], train_auc, label='Train AUC points')
plt.scatter(parameters['learning_rate'], cv_auc, label='CV AUC points')
```

```
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  24 out of  24 | elapsed:   48.1s finished
```



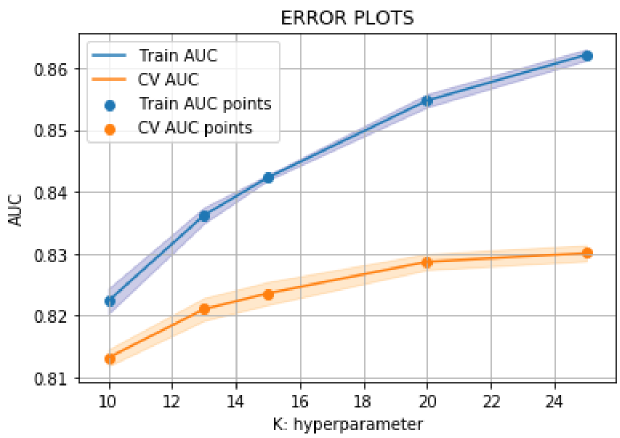
In [242]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#https://www.kaggle.com/phunter/xgboost-with-gridsearchcv
#The above link was looked at to implement xgboost with gridsearchcv
xgb_model = xgb.XGBClassifier()

#brute force scan for all parameters, here are the tricks
#usually max_depth is 6,7,8
#learning rate is around 0.05, but small changes may make big diff
#tuning min_child_weight subsample colsample_bytree can have
#much fun of fighting against overfit
#n_estimators is how many round of boosting
#finally, ensemble xgboost with multiple seeds may reduce variance
parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
              'objective':['binary:logistic'],
              'learning_rate': [0.95], #so called `eta` value
              'max_depth': [10, 13, 15, 20, 25],
              'min_child_weight': [11],
              'silent': [1],
              'subsample': [0.8],
              'colsample_bytree': [0.7],
              'n_estimators': [5], #number of trees, change it to 1000 for better results
              'missing':[-999],
              'seed': [1337]}


clf = GridSearchCV(xgb_model, parameters, n_jobs=5,
                   cv=3,
                   scoring='roc_auc',
                   verbose=2, refit=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']

plt.plot(parameters['max_depth'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['max_depth'],train_auc - train_auc_std,train_auc + train_auc_std
,alpha=0.2,color='darkblue')

plt.plot(parameters['max_depth'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
```

```
plt.gca().fill_between(parameters['max_depth'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,co
lor='darkorange')

plt.scatter(parameters['max_depth'], train_auc, label='Train AUC points')
plt.scatter(parameters['max_depth'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

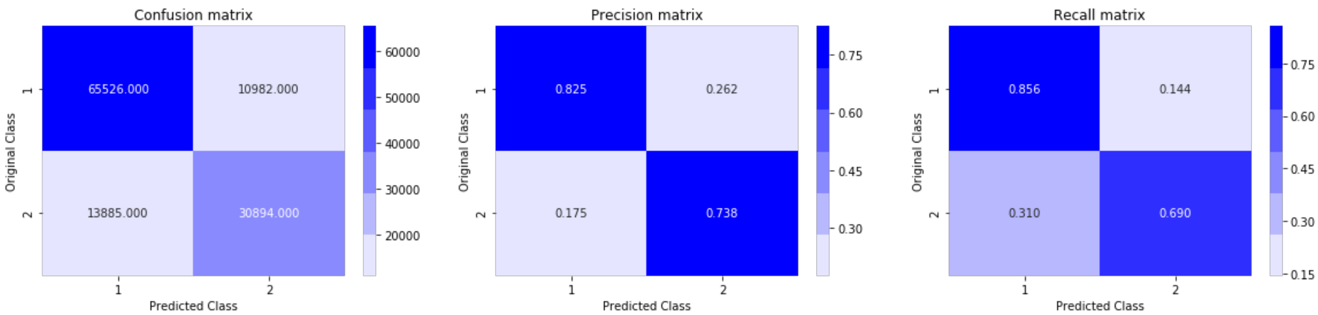
```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  15 out of  15 | elapsed:   35.3s finished
```



In [245]:

```
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
```



In [257]:

```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.5, learning_rate = 0.95,
                max_depth = 5, alpha = 10, n_estimators = 1000)
xg_reg.fit(X_train, y_train)
predict_y = xg_reg.predict(X_test)
```

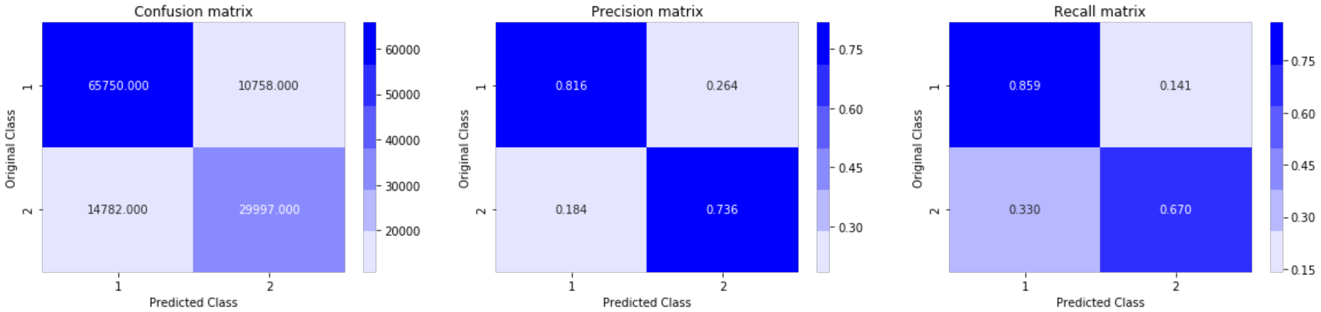
```
[22:18:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

In [258]:

```
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
```



In [ ]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ['Model', 'Train Error', 'Cross Validation Error', ]
x.add_row(["Logistic Regression", 0.418, 0.445])
x.add_row(["Linear Support Vector Machine", 0.449, 0.468])
x.add_row(["Gradient Boosting Decision Tree", 0.391, 0.412 ])
print(x)
```