

## 4.ML\_models

March 24, 2019

```
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 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 [5]: #Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 10000
    j = 0
    index_start = 1
    for df in pd.read_csv('final_features.csv', names=['Unnamed: 0', 'id', 'is_duplicate']):
        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 [2]: #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 database:")
    tables =table_names.fetchall()

```

```
print(tables[0][0])
return(len(tables))
```

```
In [3]: read_db = 'train.db'
        conn_r = create_connection(read_db)
        checkTableExists(conn_r)
        conn_r.close()
```

Tables in the database:  
data

```
In [4]: # 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 60001;", conn_r)
                conn_r.commit()
                conn_r.close()
```

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

```
Out[5]:
```

	index	Unnamed: 0	id	is_duplicate	cwc_min \
0	88097	88095.0	88095	1	0.999975000624984
1	148005	148003.0	148003	1	0.749981250468738
2	131751	131749.0	131749	1	0.999966667777741
3	9871	9869.0	9869	0	0.999975000624984
4	99079	99077.0	99077	1	0.749981250468738

	cwc_max	csc_min	csc_max	ctc_min \
0	0.999975000624984	0.833319444675922	0.714275510349852	0.899991000089999
1	0.499991666805553	0.999985714489793	0.999985714489793	0.90908264470323
2	0.999966667777741	0.499987500312492	0.499987500312492	0.714275510349852
3	0.799984000319994	0.749981250468738	0.599988000239995	0.874989062636717
4	0.374995312558593	0.249993750156246	0.199996000079998	0.499993750078124

	ctc_max	...	374_y	375_y \
0	0.818174380232907	...	-5.00604896806181	16.8518334776163
1	0.769224852116522	...	5.2621960863471	10.3345596343279
2	0.714275510349852	...	10.4113432234153	5.65417674463242
3	0.699993000069999	...	6.19151087105274	0.710954058915377
4	0.285712244912536	...	8.12903142347932	6.92441907152534

	376_y	377_y	378_y	379_y \
0	-5.37091588228941	-3.26178657100536	12.1312550008297	7.41904079914093

1	24.0024595782161	-0.733043350279331	15.7177833020687	6.26007471210323
2	10.6767095029354	-11.5996008403599	6.16212257742882	4.318210542202
3	6.83794537186623	-0.607464585453272	12.0138160921633	5.9918103441596
4	16.8435802906752	-10.6461466807523	4.99954917468131	-4.2941911816597

	380_y	381_y	382_y	383_y
0	-6.07296107709408	4.85085135698319	9.19919856637717	5.88767864555121
1	10.8977920934558	0.311331823468208	26.6197788715362	-6.09706179052591
2	-4.39765763282776	8.52527767419815	7.5825545489788	3.14942193031311
3	-7.41477456688881	5.32388159632683	0.883917570114136	9.15121423825622
4	-12.0200691521168	11.6374632790685	10.7902968525887	4.13943130895495

[5 rows x 798 columns]

```
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 \
1	0.749981250468738	0.499991666805553	0.999985714489793	0.999985714489793
2	0.999966667777741	0.999966667777741	0.499987500312492	0.499987500312492
3	0.999975000624984	0.799984000319994	0.749981250468738	0.599988000239995
4	0.749981250468738	0.374995312558593	0.249993750156246	0.199996000079998
5	0.307689940846609	0.249998437509766	0.14285510206997	0.124998437519531

	ctc_min	ctc_max	last_word_eq	first_word_eq \
1	0.90908264470323	0.769224852116522	0.0	1.0
2	0.714275510349852	0.714275510349852	1.0	1.0
3	0.874989062636717	0.699993000069999	0.0	1.0
4	0.499993750078124	0.285712244912536	1.0	0.0
5	0.238094104313789	0.192306952665567	0.0	0.0

	abs_len_diff	mean_len	...	374_y \
1	2.0	12.0	...	5.2621960863471
2	0.0	7.0	...	10.4113432234153
3	2.0	9.0	...	6.19151087105274
4	6.0	11.0	...	8.12903142347932
5	5.0	23.5	...	-12.3651023469865

	375_y	376_y	377_y	378_y \
1	10.3345596343279	24.0024595782161	-0.733043350279331	15.7177833020687
2	5.65417674463242	10.6767095029354	-11.5996008403599	6.16212257742882
3	0.710954058915377	6.83794537186623	-0.607464585453272	12.0138160921633
4	6.92441907152534	16.8435802906752	-10.6461466807523	4.99954917468131
5	16.2649692818522	21.639226064086	23.0527893770486	36.2669450156391

	379_y	380_y	381_y	382_y \
1	6.26007471210323	10.8977920934558	0.311331823468208	26.6197788715362
2	4.318210542202	-4.39765763282776	8.52527767419815	7.5825545489788
3	5.9918103441596	-7.41477456688881	5.32388159632683	0.883917570114136
4	-4.2941911816597	-12.0200691521168	11.6374632790685	10.7902968525887
5	-26.5054575204849	-1.08090314269066	9.41618685470894	-2.49915977194905

	383_y
1	-6.09706179052591
2	3.14942193031311
3	9.15121423825622
4	4.13943130895495
5	3.66466497443616

[5 rows x 794 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 numeric 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  
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 [9]: # 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 [10]: X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

```
In [11]: 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 : (42000, 794)

Number of data points in test data : (18000, 794)

```
In [12]: 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: 0.6306428571428572 Class 1: 0.3693571428571429

----- Distribution of output variable in train data -----

Class 0: 0.36933333333333335 Class 1: 0.36933333333333335

```
In [13]: # 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 as 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 corresponds to columns and axis=1 corresponds to rows in C  
    # C.sum(axis = 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 corresponds to columns and axis=1 corresponds to rows in
# C.sum(axis = 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 [14]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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))

```

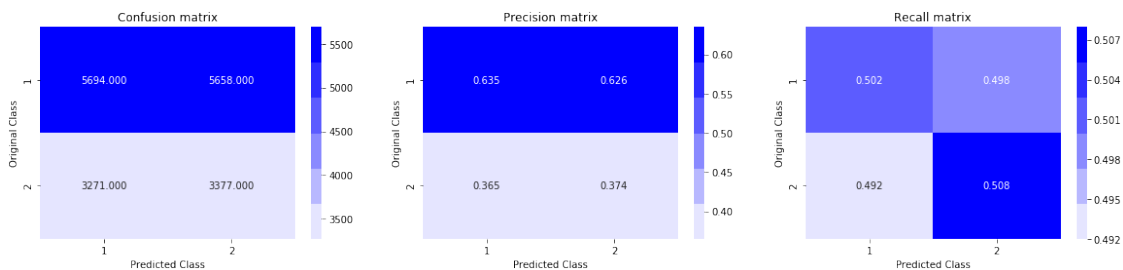
```

for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-

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

```

Log loss on Test Data using Random Model 0.8858467472564963



#### 4.4 Logistic Regression with hyperparameter tuning

```

In [17]: from tqdm.auto import tqdm
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/
# -----
# 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=opt
# 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 Gr
# predict(X)          Predict class labels for samples in X.

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

log_error_array=[]
for i in tqdm(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,

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()

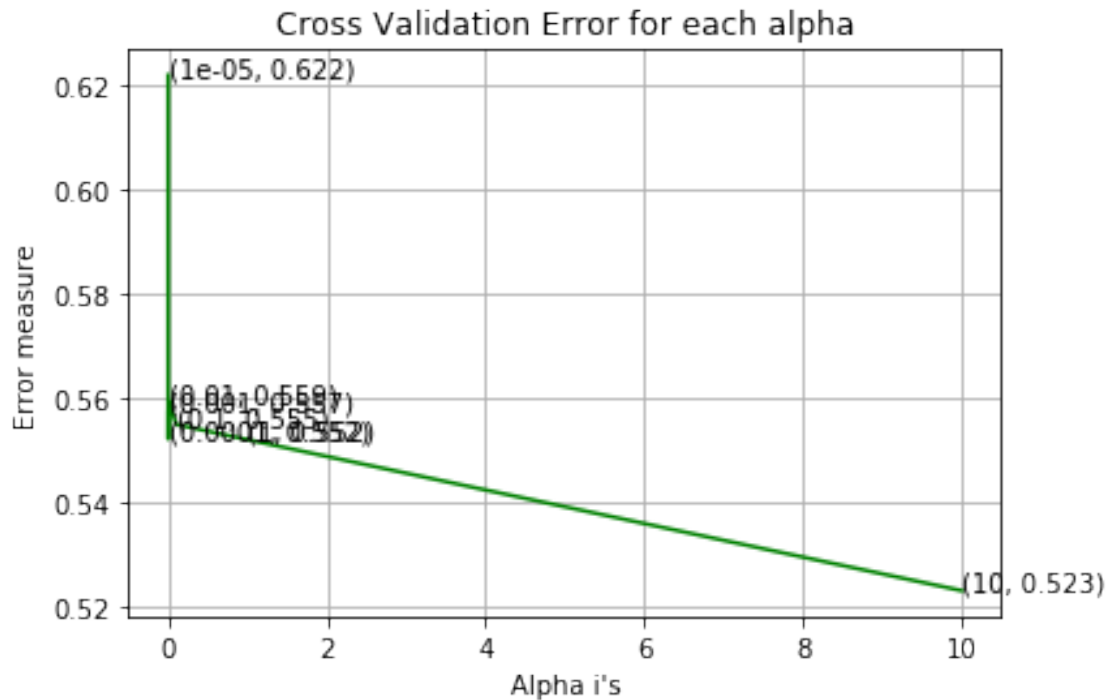
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

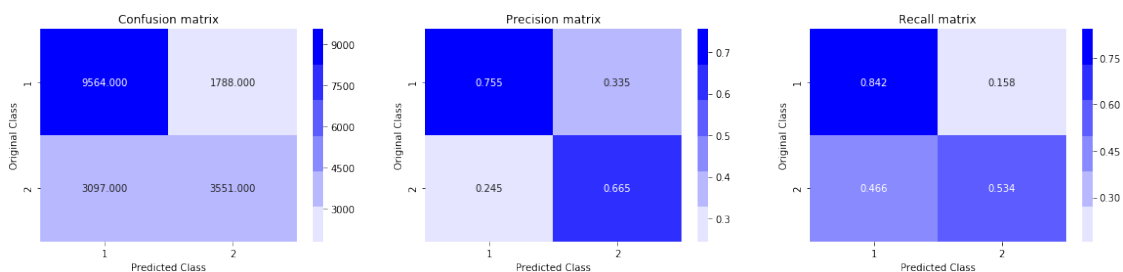
HBox(children=(IntProgress(value=0, max=7), HTML(value='')))

For values of alpha = 1e-05 The log loss is: 0.6220434819250116
For values of alpha = 0.0001 The log loss is: 0.5520418910381882
For values of alpha = 0.001 The log loss is: 0.5573915480798847
For values of alpha = 0.01 The log loss is: 0.5593118711520558
For values of alpha = 0.1 The log loss is: 0.5548563446875912
For values of alpha = 1 The log loss is: 0.5519260464527921
For values of alpha = 10 The log loss is: 0.5228888292938616

```



For values of best alpha = 10 The train log loss is: 0.5139749163644576  
 For values of best alpha = 10 The test log loss is: 0.5228888292938616  
 Total number of data points : 18000



#### 4.5 Linear SVM with hyperparameter tuning

In [19]: `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/
# -----
# 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=op
```

```

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

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

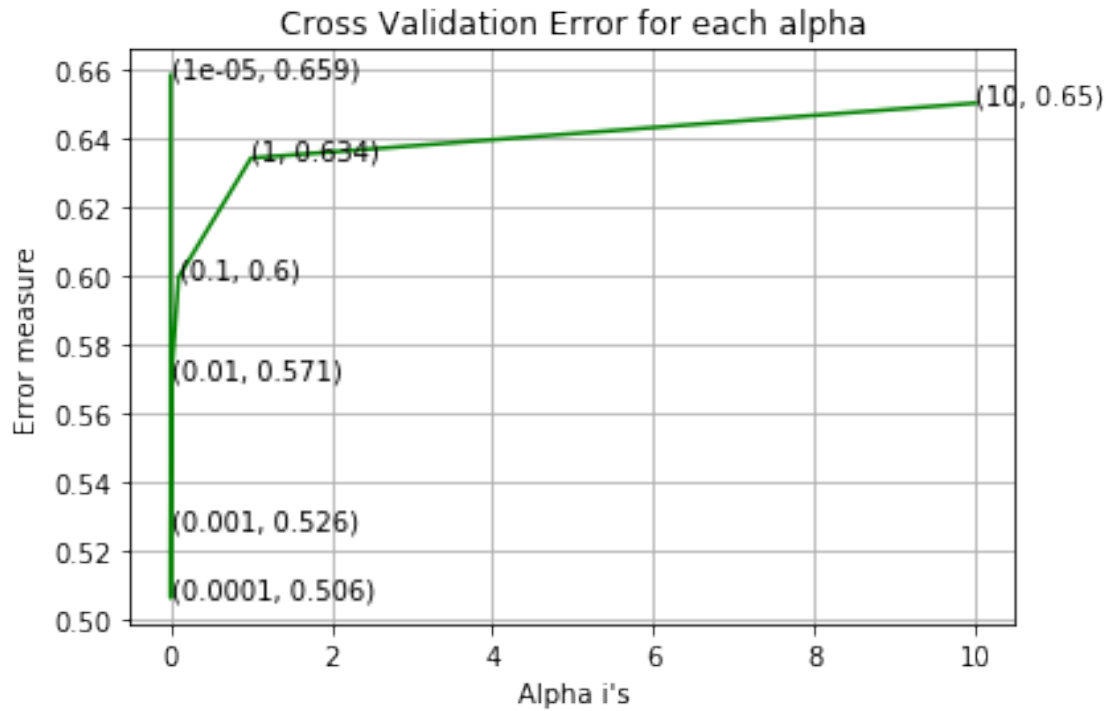
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

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, 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)

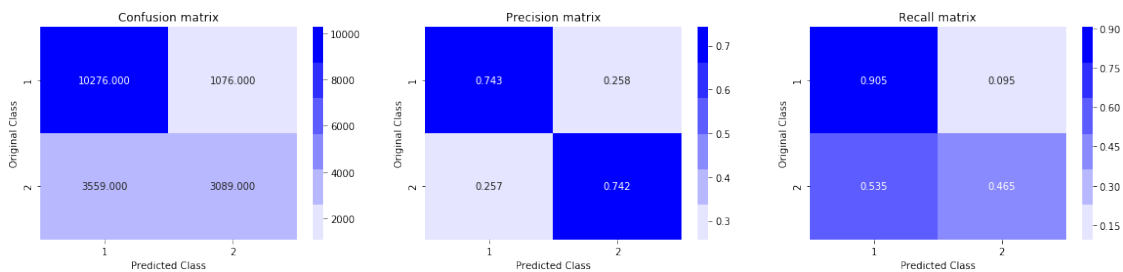
HBox(children=(IntProgress(value=0, max=7), HTML(value='')))

```

For values of alpha = 1e-05 The log loss is: 0.6585999197746482  
 For values of alpha = 0.0001 The log loss is: 0.5063896242672866  
 For values of alpha = 0.001 The log loss is: 0.5264735997478815  
 For values of alpha = 0.01 The log loss is: 0.5707068580328858  
 For values of alpha = 0.1 The log loss is: 0.5998146952530511  
 For values of alpha = 1 The log loss is: 0.6344824538860115  
 For values of alpha = 10 The log loss is: 0.6504648900469024



For values of best alpha = 0.0001 The train log loss is: 0.49823810184033107  
 For values of best alpha = 0.0001 The test log loss is: 0.5063896242672866  
 Total number of data points : 18000



## 4.6 XGBoost

```
In [20]: 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)

         xgdmatrix = 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-10))
```

[19:30:29] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[0] train-logloss:0.684636 valid-logloss:0.684557  
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.

[19:30:30] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:31] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:31] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:32] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:33] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[10] train-logloss:0.614818 valid-logloss:0.614144  
[19:30:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:44] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:45] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:46] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[19:30:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error  
[20] train-logloss:0.5635 valid-logloss:0.562454  
[19:30:48] C:\Users\Administrator\Desktop\xgboost\src\tree\updater\_prune.cc:74: tree pruning error

















[illegible]

```

[19:37:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:44] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:45] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:48] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:49] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:50] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:52] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[380]      train-logloss:0.339683      valid-logloss:0.352928
[19:37:53] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:54] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:56] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:57] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:58] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:37:59] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:01] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:03] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:04] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[390]      train-logloss:0.338474      valid-logloss:0.352421
[19:38:05] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:06] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:07] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:09] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:10] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:11] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:12] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:14] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[19:38:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning error
[399]      train-logloss:0.337379      valid-logloss:0.351811
The test log loss is: 0.3518105491030114

```

```

In [21]: 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 : 18000

