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.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



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

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

```
/home/vinay/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection
module into which all the refactored classes and functions are moved. Also note that the interface
of the new CV iterators are different from that of this module. This module will be removed in 0.2
0.
  "This module will be removed in 0.20.", DeprecationWarning)
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [0]:

```
#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','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'], 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 [0]:

```
#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 [0]:

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

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

In [0]:

```
# 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 [0]:

```
# 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 [0]:

```
data.head()
```

Out[0]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_ma
1	0.199996000079998	0.166663888935184	0.0	0.0	0.14285510206997	0.09999900000999
2	0.399992000159997	0.399992000159997	0.499987500312492	0.499987500312492	0.444439506227709	0.44443950622770
3	0.833319444675922	0.714275510349852	0.999983333611106	0.857130612419823	0.687495703151855	0.68749570315185

4	0.0 cwc_min	0.0 cwc_max	0.599988000239995 csc_min	0.499991666805553 csc_max	0.249997916684028 ctc_min	0.230767455634957 ctc_ma
5	0.749981250468738	0.749981250468738	0.499987500312492	0.499987500312492	0.624992187597655	0.624992187597655

5 rows × 794 columns

mine starts

In [4]:

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

In [5]:

```
len(data)
```

Out[5]:

```
404290
```

In [6]:

```
data.head(3)
```

Out[6]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	fi
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983	...	0.785709	0.0	1
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988	...	0.466664	0.0	1
2	2	5	6	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0	0.399992	0.333328	0.399992	0.249997	...	0.285712	0.0	1

3 rows × 21 columns

In [7]:

```
from sklearn.decomposition import TruncatedSVD
```

In [8]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [9]:

```
sample = data.iloc[list(np.random.permutation(data.index)[0:100000])]
```

In [10]:

```
len(sample)
```

Out[10]:

```
100000
```

In [12]:

```
tfidfq1 = TfidfVectorizer(max_features = 10000,max_df = 0.5)
```

In [16]:

```
sample['question1'].isnull().sum()
```

Out[16]:

```
4
```

In [17]:

```
sample['question2'].isnull().sum()
```

Out[17]:

```
1
```

In [18]:

```
sample.dropna(inplace = True)
```

In [19]:

```
sample['question2'].isnull().sum()
```

Out[19]:

```
0
```

In [20]:

```
tf_q1 = tfidfq1.fit_transform(sample['question1'])
```

In [21]:

```
tfidfq2 = TfidfVectorizer(max_features = 10000,max_df = 0.5)
tf_q2 = tfidfq2.fit_transform(sample['question2'])
```

In [23]:

```
tf_q2.shape,tf_q1.shape
```

Out[23]:

```
((99995, 10000), (99995, 10000))
```

In [24]:

```
reducer = TruncatedSVD(n_components = 150)
q1 = reducer.fit_transform(tf_q1)
```

In [25]:

```
reducer2 = TruncatedSVD(n_components = 150)
q2 = reducer2.fit_transform(tf_q2)
```

In [26]:

```
sample.head(2)
```

Out[26]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max
102019	102019	1183	23718	do you think time travel is possible	will time travel be possible in the next 10 ye...	1	0.749981	0.499992	0.000000	0.000000	...	0.299997
120021	120021	194746	194747	which are the good coaching centres for ugc ...	what is the best coaching centre for ugc net ...	1	0.666656	0.571420	0.599988	0.599988	...	0.583328

2 rows × 21 columns

In [31]:

```
len(q1)
```

Out[31]:

```
99995
```

In [33]:

```
q1 = pd.DataFrame(q1)
q1.head(2)
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...	140	14
0	0.173938	0.127102	-0.078941	0.232715	-0.051007	0.162865	-0.066427	-0.012668	-0.016599	0.014642	...	0.018382	-0.01398
1	0.146185	-0.042481	0.088168	-0.117754	-0.051930	0.059705	0.067188	0.058471	-0.000824	0.110832	...	0.010699	0.02009

2 rows × 150 columns

In [34]:

```
q1.rename(columns=lambda x: str(x)+'q1',inplace = True)
q1.head(2)
```

Out[34]:

	0q1	1q1	2q1	3q1	4q1	5q1	6q1	7q1	8q1	9q1	...	140q1	141q
0	0.173938	0.127102	-0.078941	0.232715	-0.051007	0.162865	-0.066427	-0.012668	-0.016599	0.014642	...	0.018382	-0.01398
1	0.146185	-0.042481	0.088168	-0.117754	-0.051930	0.059705	0.067188	0.058471	-0.000824	0.110832	...	0.010699	0.02009

2 rows × 150 columns

In [42]:

```python
# # q2 = pd.DataFrame(q2)
# q1.rename(columns=lambda x: str(x)+'q1',inplace = True)

q2.rename(columns=lambda x: str(x)+'q2',inplace = True)
q2.head(2)
```

Out[42]:

	0q2	1q2	2q2	3q2	4q2	5q2	6q2	7q2	8q2	9q2	...	140q2	141q2
0	0.138256	0.003171	-0.023662	-0.060445	0.015126	0.088609	0.100082	-0.147232	0.034028	0.055449	...	0.054872	-0.04668
1	0.197933	-0.105279	-0.056647	-0.076201	-0.007026	-0.073883	0.085272	0.055020	-0.050423	0.078167	...	-0.006137	0.02857

2 rows × 150 columns

In [44]:

```python
q1.rename(columns = lambda x: x[:3],inplace = True)
q1.head(2)
```

Out[44]:

	0q1	1q1	2q1	3q1	4q1	5q1	6q1	7q1	8q1	9q1	...	140	14
0	0.173938	0.127102	-0.078941	0.232715	-0.051007	0.162865	-0.066427	-0.012668	-0.016599	0.014642	...	0.018382	-0.01398
1	0.146185	-0.042481	0.088168	-0.117754	-0.051930	0.059705	0.067188	0.058471	-0.000824	0.110832	...	0.010699	0.02009

2 rows × 150 columns

In [28]:

```python
sample.index = np.arange(0,len(sample))
```

In [29]:

```python
sample.head(2)
```

Out[29]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_w
0	102019	1183	23718	do you think time travel is possible	will time travel be possible in the next 10 ye...	1	0.749981	0.499992	0.000000	0.000000	...	0.299997	0.0
1	120021	194746	194747	which are the good coaching centres for ugc ...	what is the best coaching centre for ugc net ...	1	0.666656	0.571420	0.599988	0.599988	...	0.583328	1.0

2 rows × 21 columns

In [37]:

```python
sample.drop(columns = ['id','qid1','qid2','question1','question2'],inplace = True)
```

In [40]:

```
label = sample['is_duplicate']
```

In [41]:

```
sample.drop(columns = ['is_duplicate'],inplace = True)
sample.head(2)
```

Out[41]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	token_
0	0.749981	0.499992	0.000000	0.000000	0.428565	0.299997	0.0	0.0	3.0	8.5	71
1	0.666656	0.571420	0.599988	0.599988	0.636358	0.583328	1.0	0.0	1.0	11.5	81

In [45]:

```
data = pd.concat([sample,q1,q2],axis = 1)
```

In [47]:

```
data.head(2)
```

Out[47]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	...	
0	0.749981	0.499992	0.000000	0.000000	0.428565	0.299997	0.0	0.0	3.0	8.5	...	0.0
1	0.666656	0.571420	0.599988	0.599988	0.636358	0.583328	1.0	0.0	1.0	11.5	...	-0.0

2 rows × 315 columns

4.3 Random train test split(70:30)

In [49]:

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

In [50]:

```
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 : (69996, 315)
Number of data points in test data : (29999, 315)
```

In [51]:

```
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:  0.6318361049202812 Class 1:  0.36816389507971886
---------- Distribution of output variable in train data ----------
Class 0:  0.3681789392979766 Class 1:  0.3681789392979766
```

In [52]:

```
# 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()
```

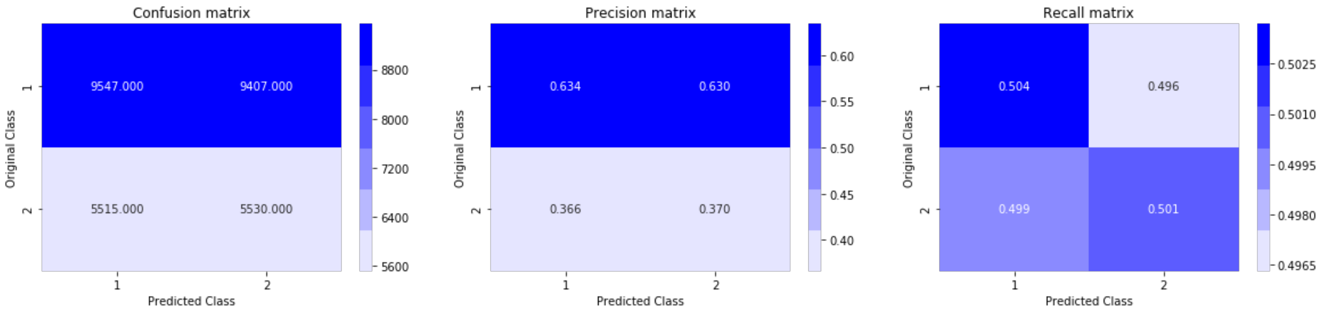
4.4 Building a random model (Finding worst-case log-loss)

In [53]:

```
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 0.884694812207
```

Log loss on Test Data using Random Model 0.884094812267

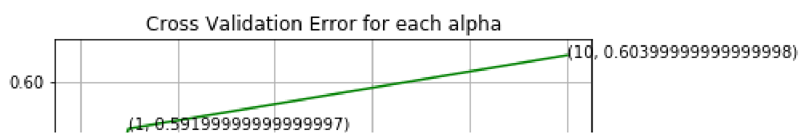


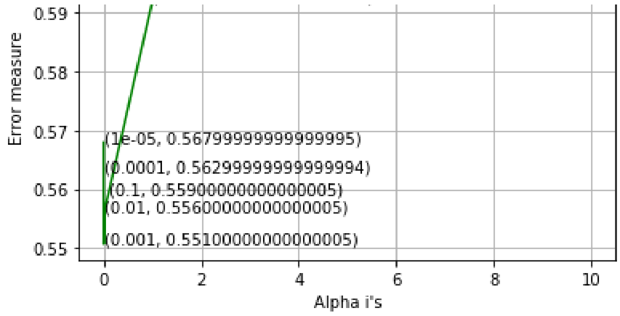
4.4 Logistic Regression with hyperparameter tuning

In [54]:

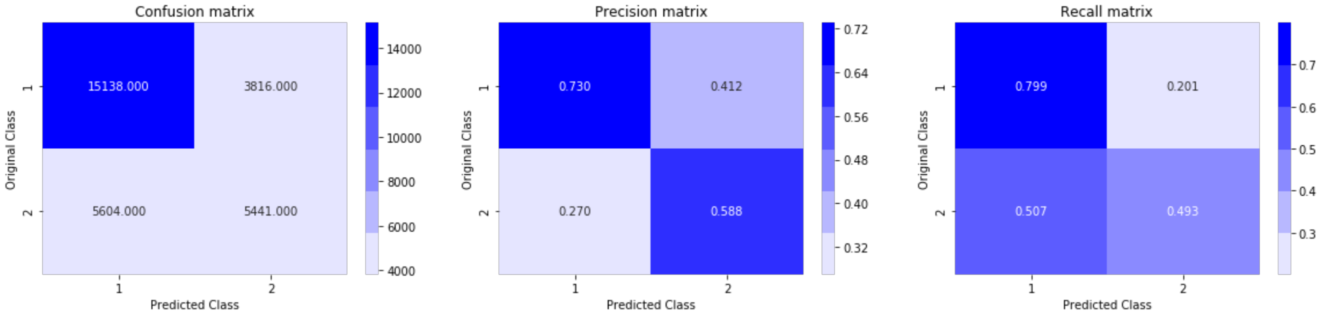
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.



log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, 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: 0.567942041588
For values of alpha =  0.0001 The log loss is: 0.562880634732
For values of alpha =  0.001 The log loss is: 0.550671919405
For values of alpha =  0.01 The log loss is: 0.556310357167
For values of alpha =  0.1 The log loss is: 0.559296228388
For values of alpha =  1 The log loss is: 0.591966293994
For values of alpha =  10 The log loss is: 0.604434861461
```






```
For values of best alpha =  0.001 The train log loss is: 0.545726262775
For values of best alpha =  0.001 The test log loss is: 0.550671919405
Total number of data points : 29999
```



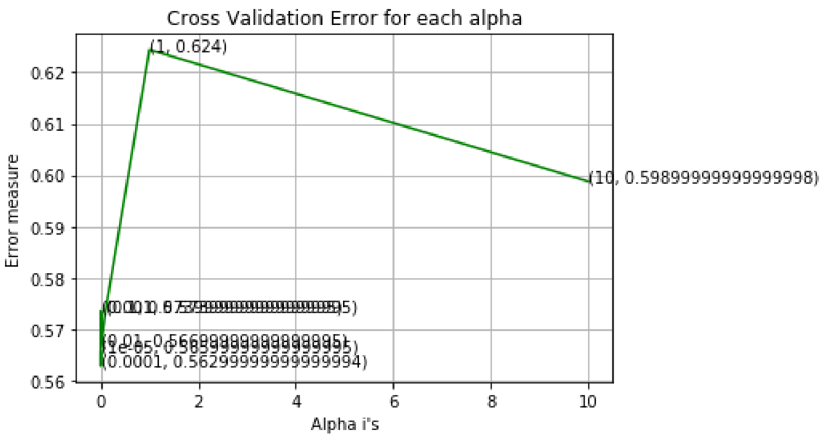
4.5 Linear SVM with hyperparameter tuning

In [55]:

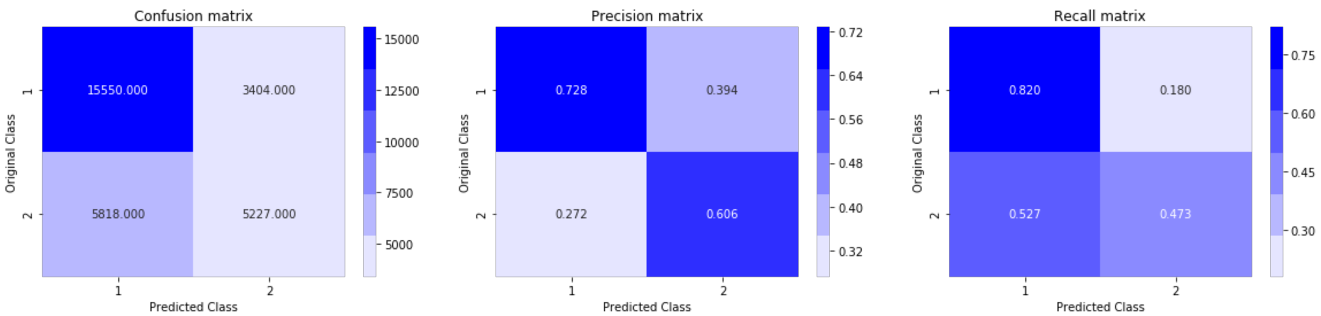
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/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, 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: 0.565592731162
For values of alpha =  0.0001 The log loss is: 0.562854712668
For values of alpha =  0.001 The log loss is: 0.573594999639
For values of alpha =  0.01 The log loss is: 0.566875517852
For values of alpha =  0.1 The log loss is: 0.573572794387
For values of alpha =  1 The log loss is: 0.624413359636
For values of alpha =  10 The log loss is: 0.598780991957
```




```
For values of best alpha =  0.0001 The train log loss is: 0.559866195184
For values of best alpha =  0.0001 The test log loss is: 0.562854712668
Total number of data points : 29999
```



4.6 XGBoost

In [69]:

```
from sklearn.externals import joblib
joblib.dump(sig_clf, "svd_quora.clf")
```

Out[69]:

```
['svd_quora.clf']
```

In [63]:

```
np.save('X_train.npy',X_train)
```

In [66]:

```
np.save('y_train.npy',y_train)
np.save('y_test.npy',y_test)
np.save('X_test.npy',X_test)
```

In [2]:

```
X_train = np.load('X_train.npy')
y_train = np.load('y_train.npy')
y_test = np.load('y_test.npy')
X_test = np.load('X_test.npy')
```

In [3]:

```
X_train.shape
```

Out[3]:

```
(69996, 315)
```

In [58]:

```
X_train.shape,len(y_train)
```

Out[58]:

```
((69996, 315), 69996)
```

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
xgb = xgb.XGBClassifier(learning_rate=0.02, n_estimators=600, objective='binary:logistic',
                    silent=True,nthread=1)


params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }



    # A parameter grid for XGBoost


random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=20, scoring='accuracy',
n_jobs=3, cv=3, verbose=3, random_state=1001 )
random_search.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[CV] subsample=1.0, min_child_weight=5, max_depth=3, gamma=5, colsample_bytree=1.0
[CV] subsample=1.0, min_child_weight=5, max_depth=3, gamma=5, colsample_bytree=1.0
[CV] subsample=1.0, min_child_weight=5, max_depth=3, gamma=5, colsample_bytree=1.0
[CV]  subsample=1.0, min_child_weight=5, max_depth=3, gamma=5, colsample_bytree=1.0,
score=0.7733156180353163, total= 8.4min
[CV] subsample=0.6, min_child_weight=1, max_depth=5, gamma=1.5, colsample_bytree=0.8
[CV]  subsample=1.0, min_child_weight=5, max_depth=3, gamma=5, colsample_bytree=1.0,
score=0.7763157894736842, total= 8.5min
[CV] subsample=0.6, min_child_weight=1, max_depth=5, gamma=1.5, colsample_bytree=0.8
[CV]  subsample=1.0, min_child_weight=5, max_depth=3, gamma=5, colsample_bytree=1.0,
score=0.7736584947711298, total= 8.5min
```

```
[CV] subsample=0.6, min_child_weight=1, max_depth=5, gamma=1.5, colsample_bytree=0.8
[CV]  subsample=0.6, min_child_weight=1, max_depth=5, gamma=1.5, colsample_bytree=0.8,
score=0.7948739927995886, total=10.5min
[CV] subsample=0.8, min_child_weight=5, max_depth=5, gamma=1, colsample_bytree=0.8
[CV]  subsample=0.6, min_child_weight=1, max_depth=5, gamma=1.5, colsample_bytree=0.8,
score=0.792902451568661, total=10.6min
[CV] subsample=0.8, min_child_weight=5, max_depth=5, gamma=1, colsample_bytree=0.8
[CV]  subsample=0.6, min_child_weight=1, max_depth=5, gamma=1.5, colsample_bytree=0.8,
score=0.7919166809531973, total=10.7min
[CV] subsample=0.8, min_child_weight=5, max_depth=5, gamma=1, colsample_bytree=0.8
[CV]  subsample=0.8, min_child_weight=5, max_depth=5, gamma=1, colsample_bytree=0.8,
score=0.7918738213612206, total=11.2min
[CV] subsample=1.0, min_child_weight=5, max_depth=5, gamma=5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=5, max_depth=5, gamma=1, colsample_bytree=0.8,
score=0.7940168009600549, total=11.2min
[CV] subsample=1.0, min_child_weight=5, max_depth=5, gamma=5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=5, max_depth=5, gamma=1, colsample_bytree=0.8,
score=0.7914023658494771, total=11.2min
[CV] subsample=1.0, min_child_weight=5, max_depth=5, gamma=5, colsample_bytree=0.6
[CV]  subsample=1.0, min_child_weight=5, max_depth=5, gamma=5, colsample_bytree=0.6,
score=0.7924309960569176, total= 8.9min
[CV] subsample=0.8, min_child_weight=1, max_depth=4, gamma=1, colsample_bytree=1.0
[CV]  subsample=1.0, min_child_weight=5, max_depth=5, gamma=5, colsample_bytree=0.6,
score=0.7918738213612206, total= 9.0min
[CV] subsample=0.8, min_child_weight=1, max_depth=4, gamma=1, colsample_bytree=1.0
[CV]  subsample=1.0, min_child_weight=5, max_depth=5, gamma=5, colsample_bytree=0.6,
score=0.7914880850334305, total= 9.0min
[CV] subsample=0.8, min_child_weight=1, max_depth=4, gamma=1, colsample_bytree=1.0
[CV]  subsample=0.8, min_child_weight=1, max_depth=4, gamma=1, colsample_bytree=1.0,
score=0.7851877250128578, total=10.9min
[CV] subsample=1.0, min_child_weight=10, max_depth=4, gamma=1.5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=1, max_depth=4, gamma=1, colsample_bytree=1.0,
score=0.7864735127721584, total=10.9min
[CV] subsample=1.0, min_child_weight=10, max_depth=4, gamma=1.5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=1, max_depth=4, gamma=1, colsample_bytree=1.0,
score=0.7851020058289045, total=10.9min
[CV] subsample=1.0, min_child_weight=10, max_depth=4, gamma=1.5, colsample_bytree=0.6
[CV]  subsample=1.0, min_child_weight=10, max_depth=4, gamma=1.5, colsample_bytree=0.6,
score=0.7856163209326247, total= 7.0min
[CV] subsample=1.0, min_child_weight=1, max_depth=5, gamma=5, colsample_bytree=0.6
[CV]  subsample=1.0, min_child_weight=10, max_depth=4, gamma=1.5, colsample_bytree=0.6,
score=0.7844162523572775, total= 7.1min
[CV] subsample=1.0, min_child_weight=1, max_depth=5, gamma=5, colsample_bytree=0.6
[CV]  subsample=1.0, min_child_weight=10, max_depth=4, gamma=1.5, colsample_bytree=0.6,
score=0.7836019201097205, total= 7.0min
[CV] subsample=1.0, min_child_weight=1, max_depth=5, gamma=5, colsample_bytree=0.6
[CV]  subsample=1.0, min_child_weight=1, max_depth=5, gamma=5, colsample_bytree=0.6,
score=0.7929881707526144, total= 9.2min
[CV] subsample=0.8, min_child_weight=1, max_depth=3, gamma=2, colsample_bytree=0.8
[CV]  subsample=1.0, min_child_weight=1, max_depth=5, gamma=5, colsample_bytree=0.6,
score=0.792645294016801, total= 9.2min
[CV] subsample=0.8, min_child_weight=1, max_depth=3, gamma=2, colsample_bytree=0.8
[CV]  subsample=1.0, min_child_weight=1, max_depth=5, gamma=5, colsample_bytree=0.6,
score=0.789516543802503, total= 9.2min
[CV] subsample=0.8, min_child_weight=1, max_depth=3, gamma=2, colsample_bytree=0.8
[CV]  subsample=0.8, min_child_weight=1, max_depth=3, gamma=2, colsample_bytree=0.8,
score=0.773358477627293, total= 6.6min
[CV]  subsample=0.8, min_child_weight=1, max_depth=3, gamma=2, colsample_bytree=0.8,
score=0.7796159780558889, total= 6.6min
[CV] subsample=0.8, min_child_weight=1, max_depth=5, gamma=0.5, colsample_bytree=0.6
[CV] subsample=0.8, min_child_weight=1, max_depth=5, gamma=0.5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=1, max_depth=3, gamma=2, colsample_bytree=0.8,
score=0.7747299845705469, total= 6.7min
[CV] subsample=0.8, min_child_weight=1, max_depth=5, gamma=0.5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=1, max_depth=5, gamma=0.5, colsample_bytree=0.6,
score=0.7933310474884279, total= 8.9min
[CV] subsample=0.8, min_child_weight=1, max_depth=4, gamma=1.5, colsample_bytree=0.6
[CV]  subsample=0.8, min_child_weight=1, max_depth=5, gamma=0.5, colsample_bytree=0.6,
score=0.7929453111606377, total= 8.9min
[CV] subsample=0.8, min_child_weight=1, max_depth=4, gamma=1.5, colsample_bytree=0.6
```

```
[Parallel(n_jobs=3)]: Done  26 tasks      | elapsed: 82.1min
```

```
[CV]  subsample=0.8, min_child_weight=1, max_depth=5, gamma=0.5, colsample_bytree=0.6,
score=0.7925167152408709, total= 8.9min
[CV] subsample=0.8, min_child_weight=1, max_depth=4, gamma=1.5, colsample_bytree=0.6
```

In [0]:

```
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
```