In [2]:

```python
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

import spacy

from tqdm import tqdm
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: DeprecationWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

4. Machine Learning Models

In [0]:

```python
# from google.colab import drive
# drive.mount('/content/drive')
```

In [0]:

```python
# cd DONOR_CHOOSE_KNN/quora
```

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('../input/quora/Quora/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]:

```
# # final_data = pd.read_csv('final_features.csv')
# import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))
```

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()
```

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 [7]:

```
# data = pd.read_csv('../input/quora/Quora/final_features.csv')

# data.head(5)

nlp_data = pd.read_csv('nlp_features_train.csv', encoding = "latin-1")
preprocessed_data = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding = "latin-1")
preprocessed_data = preprocessed_data.drop(['qid1', 'qid2', 'question1', 'question2', 'is_duplicate'],axis=1)
# df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
# df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)

# df3_q1['id']=df1['id']
# df3_q2['id']=df1['id']
# df1  = df1.merge(df2, on='id',how='left')
# df2  = df3_q1.merge(df3_q2, on='id',how='left')
# result  = df1.merge(df2, on='id',how='left')
# df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
# df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)
# df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
print(nlp_data.columns)
print(preprocessed_data.columns)
print("preprocessed_data shape =",preprocessed_data.shape)
print("nlp_data shape = ",nlp_data.shape)
```

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', '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'],
      dtype='object')
Index(['id', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share', 'freq_q1+q2',
       'freq_q1-q2'],
      dtype='object')
preprocessed_data shape = (100000, 12)
nlp_data shape =  (100000, 21)
```

In [8]:

```python
nan_rows = nlp_data[nlp_data.isnull().any(1)]
print("Number of null entries = ", nan_rows.shape)

clean_nlp_data = nlp_data.dropna(axis = 0, how ='any')
print("Number of entries left after droping the null entries = ",clean_nlp_data.shape[0])
```

```
Number of null entries =  (6, 21)
Number of entries left after droping the null entries =  99994
```

In [0]:

```python
data = clean_nlp_data.merge(preprocessed_data, on="id", how="left")

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

In [10]:

```python
nan_rows = data[data.isnull().any(1)]
print("Number of null entries = ", nan_rows.shape)

data.head(5)
# print(type(data['cwc_min']))
# print(data['cwc_min'])
# data['cwc_min'].apply(pd.to_numeric)
#
```

```
Number of null entries =  (0, 31)
```

Out[10]:

	id	qid1	qid2	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	al
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...	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	
1	1	3	4	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	
2	2	5	6	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	
3	3	7	8	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	
4	4	9	10	which one dissolve in water quikly sugar	which fish would survive in salt water	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	

id	qid1	qid2	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	ab

salt...

4.2 Converting strings to numerics

In [0]:

```
# 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:
#     print(i)
#     data[i] = data[i].apply(pd.to_numeric)
#     print(i)
```

In [0]:

```
# 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 [0]:

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

In [12]:

```
print(type(X_train))
print(type(X_test))
print(type(y_train))
print(type(y_test))
print("=="*50)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
====================================================================================================
(69995, 31)
(29999, 31)
(69995,)
(29999,)
```

Tfidf weighted W2Vec

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(X_train['question1']) + list(X_train['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions)

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

FOR TRAIN DATA

In [14]:

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

# FOR TRAIN DATA

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar

for qu1 in tqdm(list(X_train['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
X_train['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 69995/69995 [11:38<00:00, 100.26it/s]
```

In [15]:

```
X_train['q1_feats_m'].head(5)
```

Out[15]:

```
77387    [42.92952561378479, -78.06101938523352, 37.328...
34262    [66.40157455205917, -62.90219736099243, -3.168...
83765    [154.74325448274612, -31.549048513174057, -54....
76827    [68.17155885696411, -59.59435647726059, -51.83...
94493    [118.61638343334198, -199.699702501297, -73.87...
Name: q1_feats_m, dtype: object
```

In [16]:

```
vecs2 = []
for qu2 in tqdm(list(X_train['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
X_train['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 69995/69995 [11:41<00:00, 99.80it/s]
```

In [0]:

```
# df3_q1 = pd.DataFrame(X_train.q1_feats_m.values.tolist(), index= X_train.index)
# df3_q2 = pd.DataFrame(X_train.q2_feats_m.values.tolist(), index= X_train.index)
```

In [18]:

```
X_train['q2_feats_m'].head(5)
```

Out[18]:

```
77387    [89.03142619132996, -66.80487707257271, 18.497...
34262    [-58.02090382575989, 10.582061052322388, -93.9...
83765    [238.44520664215088, 53.86913478374481, -70.78...
76827    [80.08975768089294, -58.981304690241814, -42.7...
94493    [220.2964512705803, -137.26451462507248, -85.9...
Name: q2_feats_m, dtype: object
```

FOR TEST DATA

In [19]:

```
# FOR TEST DATA

vecs1_test = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar

for qu1 in tqdm(list(X_test['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1_test.append(mean_vec1)
X_test['q1_feats_m'] = list(vecs1_test)
```

```
100%|██████████| 29999/29999 [04:58<00:00, 100.52it/s]
```

In [20]:

```
X_test['q1_feats_m'].head(5)
```

Out[20]:

```
22101    [17.958252295851707, -25.70195958018303, 24.60...
81113    [-46.77597823739052, 40.94902968406677, -82.79...
93300    [96.17097091674805, -127.29158794879913, -134....
35918    [108.213887155056, 21.325669050216675, -57.288...
94903    [101.91307735443115, -64.84906335175037, -28.6...
Name: q1_feats_m, dtype: object
```

In [21]:

```
vecs2_test = []
for qu2 in tqdm(list(X_test['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
```

```
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2_test.append(mean_vec2)
X_test['q2_feats_m'] = list(vecs2_test)
```

```
100%|██████████| 29999/29999 [04:59<00:00, 100.08it/s]
```

In [22]:

```
X_test['q2_feats_m'].head(5)
```

Out[22]:

```
22101    [309.50440019369125, -130.29062724113464, 265....
81113    [-16.74356174468994, 17.549556016921997, -54.0...
93300    [138.9835479259491, -95.82764136791229, -158.8...
35918    [66.88981437683105, -30.132676362991333, -20.2...
94903    [64.071160197258, -69.28862392902374, -61.9866...
Name: q2_feats_m, dtype: object
```

In [0]:

```
df_train = X_train.drop(['qid1','qid2','question1','question2'],axis=1)
df_train_q1 = pd.DataFrame(X_train.q1_feats_m.values.tolist(), index= df_train.index)
df_train_q2 = pd.DataFrame(X_train.q2_feats_m.values.tolist(), index= df_train.index)
```

In [24]:

```
df_train_q1.head(1)
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	10
77387	42.929526	-78.061019	37.328049	-30.88724	-52.762367	5.918803	37.552955	59.861208	-1.327656	-31.216255	-24.102007	32.1914

1 rows × 96 columns

In [0]:

```
df_train_q1['id']=X_train['id']
df_train_q2['id']=X_train['id']
# df1  = df1.merge(df2, on='id',how='left') # X_train
df2_train  = df_train_q1.merge(df_train_q2 , on='id', how='left') # df_train_q1 + df_train_q2
X_train_final  = X_train.merge(df2_train, on = 'id', how ='left')
```

In [26]:

```
nan_rows = X_train_final[X_train_final.isnull().any(1)]
print("Number of null entries = ", nan_rows.shape)
X_train_final.drop(['id','qid1','qid2','question1','question2'],axis=1,inplace=True)
X_train_final.head(5)
```

```
Number of null entries =  (0, 225)
```

Out[26]:

	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
0	0.999967	0.999967	0.999975	0.799984	0.999986	0.874989	1.0	1.0	1.0	7.5	100
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	3.0	7.5	39

	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
2	0.199998	0.142856	0.666656	0.444440	0.352939	0.222221	0.0	1.0	10.0	22.0	51
3	0.666644	0.399992	0.333328	0.249997	0.444440	0.307690	1.0	0.0	4.0	11.0	68
4	0.000000	0.000000	0.499992	0.428565	0.199999	0.166666	0.0	0.0	3.0	16.5	36

5 rows × 220 columns

In [0]:

```
X_train_finaldf_test = X_test.drop(['qid1','qid2','question1','question2'],axis=1)
df_test_q1 = pd.DataFrame(X_test.q1_feats_m.values.tolist(), index= df_test.index)
df_test_q2 = pd.DataFrame(X_test.q2_feats_m.values.tolist(), index= df_test.index)
```

In [0]:

```
df_test_q1['id']=X_test['id']
df_test_q2['id']=X_test['id']
# df1  = df1.merge(df2, on='id',how='left') # X_train
df2_test  = df_test_q1.merge(df_test_q2 , on='id', how='left') # df_train_q1 + df_train_q2
X_test_final  = X_test.merge(df2_test, on = 'id', how ='left')
```

In [0]:

```
X_test_final.drop(['q1_feats_m','q2_feats_m'],axis=1,inplace=True)
X_train_final.drop(['q1_feats_m','q2_feats_m'],axis=1,inplace=True)
```

In [59]:

```
X_train_final.head(1)
```

Out[59]:

	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
0	0.999967	0.999967	0.999975	0.799984	0.999986	0.874989	1.0	1.0	1.0	7.5	100

1 rows × 218 columns

In [30]:

```
nan_rows = X_test_final[X_test_final.isnull().any(1)]
print("Number of null entries = ", nan_rows.shape)
X_test_final.drop(['id','qid1','qid2','question1','question2'],axis=1,inplace=True)
X_test_final.head(5)
```

Number of null entries = (0, 223)

Out[30]:

	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
0	0.090908	0.076922	0.461535	0.352939	0.205882	0.189189	0.0	0.0	3.0	35.5	41
1	0.999967	0.999967	0.333322	0.333322	0.666656	0.666656	0.0	0.0	0.0	6.0	92
2	0.799992	0.799992	0.999989	0.999989	0.809520	0.739127	0.0	1.0	2.0	22.0	97
3	0.999967	0.499992	0.249994	0.166664	0.571420	0.307690	0.0	0.0	6.0	10.0	73

4	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
	0.999980	0.999980	0.999900	0.249994	0.999983	0.666659	1.0	0.0	3.0	7.5	100

5 rows × 218 columns

In [0]:

```
# print(type(X_train_final))
# print(type(X_test_final))
```

In [0]:

```
# print("Number of data points in train data :",X_train.shape)
# print("Number of data points in test data :",X_test.shape)
```

In [49]:

```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len)
print("-"*10, "Distribution of output variable in test data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6274305307521966 Class 1:  0.37256946924780343
---------- Distribution of output variable in test data ----------
Class 0:  0.3725457515250508 Class 1:  0.3725457515250508
```

In [0]:

```
# 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")
```

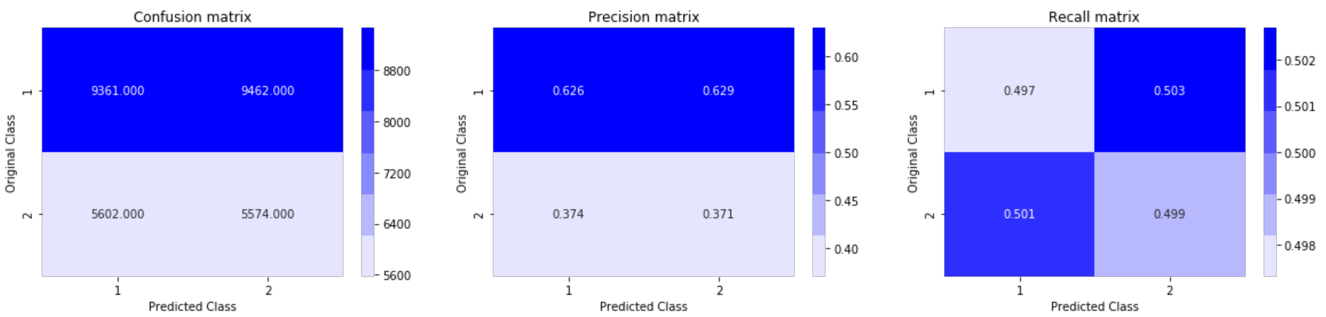
```
plt.title("Confusion matrix")

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 [51]:

```
# 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 0.8899443787399149



In [32]:

```
# y_train = y_train.tolist()
# y_test = y_test.tolist()
# type(y_train),type(y_test),type(X_train_final),type(X_test_final)
X_train_final.head(5)
```

Out[32]:

	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
0	0.999967	0.999967	0.999975	0.799984	0.999986	0.874989	1.0	1.0	1.0	7.5	100
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	3.0	7.5	39
2	0.199998	0.142856	0.666656	0.444440	0.352939	0.222221	0.0	1.0	10.0	22.0	51
3	0.666644	0.399992	0.333328	0.249997	0.444440	0.307690	1.0	0.0	4.0	11.0	68
4	0.000000	0.000000	0.499992	0.428565	0.199999	0.166666	0.0	0.0	3.0	16.5	36

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

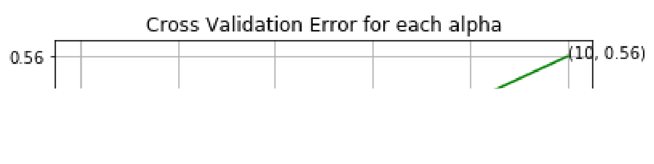
5 rows × 218 columns

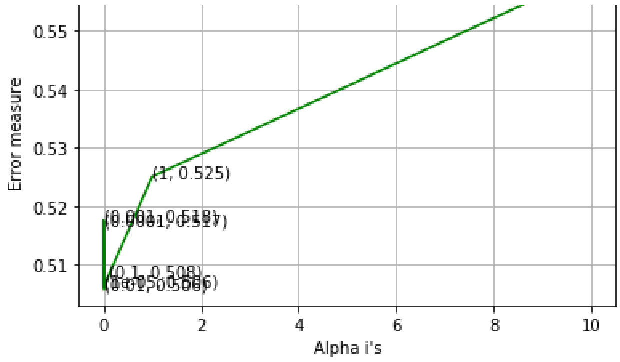
4.4 Logistic Regression with hyperparameter tuning

In [53]:

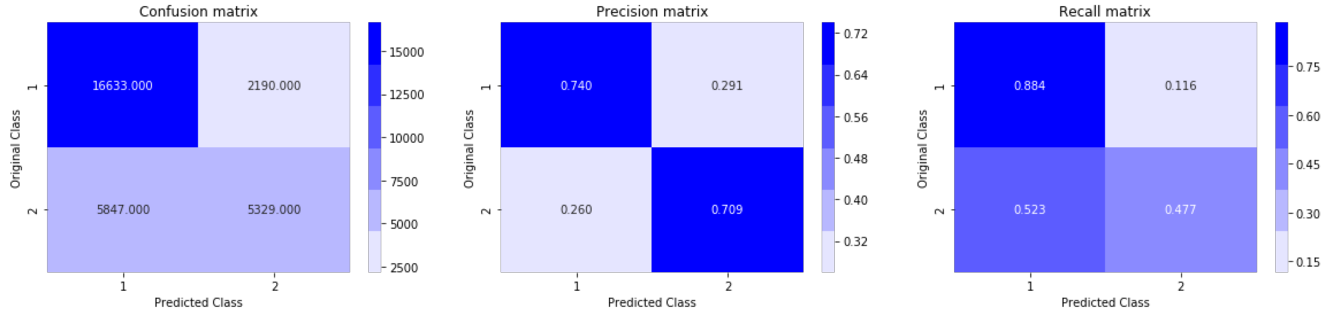
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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='l2', loss='log', random_state=42)
    clf.fit(X_train_final, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_final, y_train)
    predict_y = sig_clf.predict_proba(X_test_final)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(X_train_final)
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-1
5))
predict_y = sig_clf.predict_proba(X_test_final)
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: 0.5063727374740048
For values of alpha =  0.0001 The log loss is: 0.5167159185514054
For values of alpha =  0.001 The log loss is: 0.5176136436939562
For values of alpha =  0.01 The log loss is: 0.5057223026944383
For values of alpha =  0.1 The log loss is: 0.508105471628662
For values of alpha =  1 The log loss is: 0.5250693289796764
For values of alpha =  10 The log loss is: 0.5599229563483208
```





For values of best alpha = 0.01 The train log loss is: 0.5046804357200726
For values of best alpha = 0.01 The test log loss is: 0.5057223026944383
Total number of data points : 29999



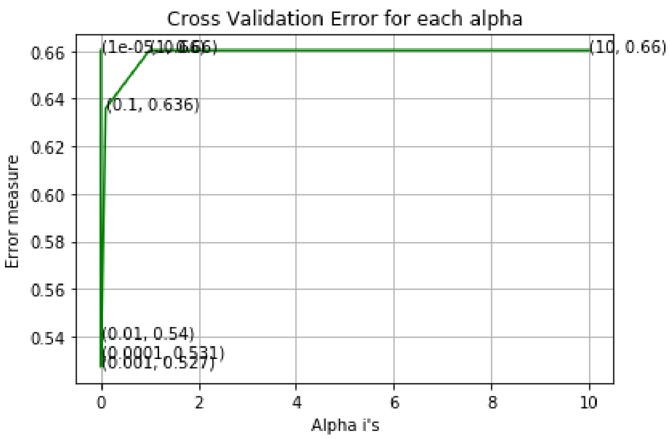
4.5 Linear SVM with hyperparameter tuning

In [54]:

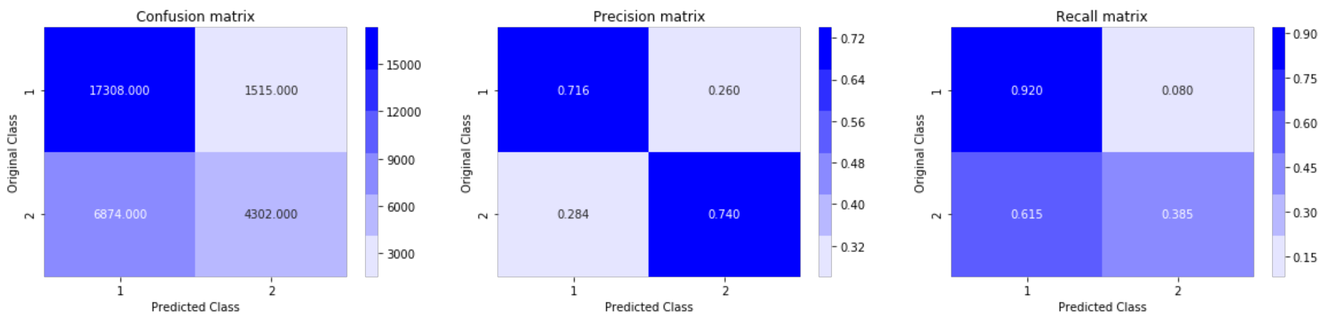
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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_final, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_final, y_train)
    predict_y = sig_clf.predict_proba(X_test_final)
    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_final, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_final, y_train)

predict_y = sig_clf.predict_proba(X_train_final)
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-1
5))
predict_y = sig_clf.predict_proba(X_test_final)
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: 0.6602966866509733
For values of alpha =  0.0001 The log loss is: 0.5314219356432783
For values of alpha =  0.001 The log loss is: 0.5273670365703893
For values of alpha =  0.01 The log loss is: 0.5397009610152068
For values of alpha =  0.1 The log loss is: 0.635886855424519
For values of alpha =  1 The log loss is: 0.6602966866509733
For values of alpha =  10 The log loss is: 0.6602966866509733
```




```
For values of best alpha =  0.001 The train log loss is: 0.5243945761542032
For values of best alpha =  0.001 The test log loss is: 0.5273670365703893
Total number of data points : 29999
```



4.6 XGBoost

In [55]:

```
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_final, label=y_train)
d_test = xgb.DMatrix(X_test_final, label=y_test)
```

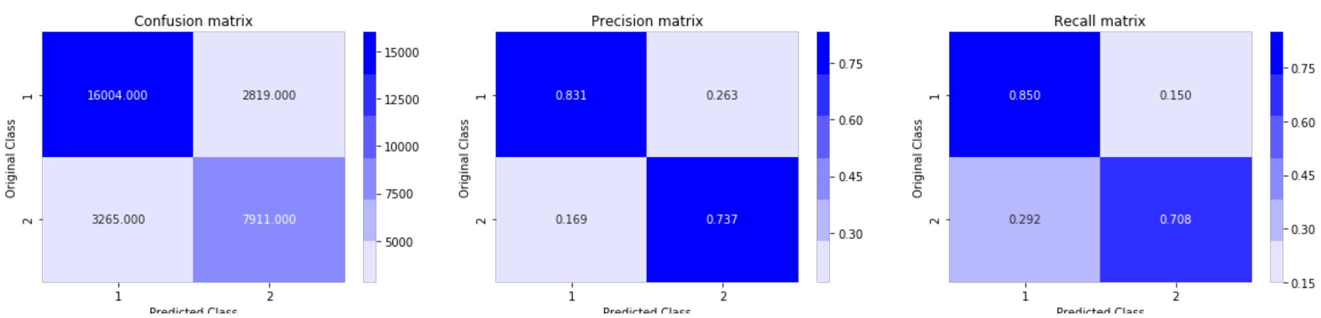
```
watchlist = [(d_train, 'train'), (d_test, 'valid')]

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

xgdmat = xgb.DMatrix(X_train_final,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:0.685498 valid-logloss:0.685682
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:0.623831 valid-logloss:0.62429
[20] train-logloss:0.578631 valid-logloss:0.579297
[30] train-logloss:0.544742 valid-logloss:0.545435
[40] train-logloss:0.518342 valid-logloss:0.519185
[50] train-logloss:0.497807 valid-logloss:0.49875
[60] train-logloss:0.481548 valid-logloss:0.482633
[70] train-logloss:0.468331 valid-logloss:0.469568
[80] train-logloss:0.457864 valid-logloss:0.459311
[90] train-logloss:0.449296 valid-logloss:0.450815
[100] train-logloss:0.442168 valid-logloss:0.443801
[110] train-logloss:0.435876 valid-logloss:0.43767
[120] train-logloss:0.430517 valid-logloss:0.432278
[130] train-logloss:0.426018 valid-logloss:0.427905
[140] train-logloss:0.422194 valid-logloss:0.424166
[150] train-logloss:0.418956 valid-logloss:0.421139
[160] train-logloss:0.416271 valid-logloss:0.418558
[170] train-logloss:0.413808 valid-logloss:0.416208
[180] train-logloss:0.411563 valid-logloss:0.414119
[190] train-logloss:0.409571 valid-logloss:0.412271
[200] train-logloss:0.407646 valid-logloss:0.410499
[210] train-logloss:0.405749 valid-logloss:0.40876
[220] train-logloss:0.404282 valid-logloss:0.40745
[230] train-logloss:0.402488 valid-logloss:0.405832
[240] train-logloss:0.400895 valid-logloss:0.404464
[250] train-logloss:0.399381 valid-logloss:0.403186
[260] train-logloss:0.397842 valid-logloss:0.401811
[270] train-logloss:0.396337 valid-logloss:0.400536
[280] train-logloss:0.394913 valid-logloss:0.39942
[290] train-logloss:0.393505 valid-logloss:0.398252
[300] train-logloss:0.3922 valid-logloss:0.397225
[310] train-logloss:0.390965 valid-logloss:0.396271
[320] train-logloss:0.389736 valid-logloss:0.395303
[330] train-logloss:0.388481 valid-logloss:0.394327
[340] train-logloss:0.387476 valid-logloss:0.39356
[350] train-logloss:0.386395 valid-logloss:0.39274
[360] train-logloss:0.385402 valid-logloss:0.391996
[370] train-logloss:0.384382 valid-logloss:0.391239
[380] train-logloss:0.383396 valid-logloss:0.390574
[390] train-logloss:0.382493 valid-logloss:0.389941
[399] train-logloss:0.381667 valid-logloss:0.389374
The test log loss is: 0.38937737766138825
```

In [56]:

```
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 : 29999



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.

Simple Tfidf

In [33]:

```
data.head(5)
```

Out[33]:

	id	qid1	qid2	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	ab
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...	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	
1	1	3	4	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	
2	2	5	6	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	
3	3	7	8	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	
4	4	9	10	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	

In [0]:

```
# # final_data.head(5)
# nlp_data = pd.read_csv('../input/quora/Quora/nlp_features_train.csv', encoding = 'latin-1')
# nlp_data.columns
```

In [0]:

```
data["question"] = data["question1"].map(str) +\
            data["question2"].map(str)
```

In [36]:

```
data.drop(['id','qid1', 'qid2', 'question1', 'question2'], axis=1, inplace=True)
data.head(5)
```

Out[36]:

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

	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
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	100
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	86
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	63
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	28
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	67

splitting of data for tfidf

In [0]:

```
X_train_tdif,X_test_tdif, y_train_tdif, y_test_tdif = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

In [40]:

```
print(X_train_tdif.shape)
print(X_test_tdif.shape)
print(y_train_tdif.shape)
print(y_test_tdif.shape)
```

```
(69995, 27)
(29999, 27)
(69995,)
(29999,)
```

TFIDF Vectorization

In [41]:

```
vectorizer_tfidf_question_1 = TfidfVectorizer()

X_train_question_tfidf = vectorizer_tfidf_question_1.fit_transform(X_train_tdif['question'])
X_test_question_tfidf = vectorizer_tfidf_question_1.transform(X_test_tdif['question'])
print("Shape of X_train_question_1_tfidf matrix ",X_train_question_tfidf.shape)
print("Shape of X_test_question_1_tfidf matrix",X_test_question_tfidf.shape)
```

```
Shape of X_train_question_1_tfidf matrix  (69995, 39173)
Shape of X_test_question_1_tfidf matrix (29999, 39173)
```

In [0]:

```
X_train_tdif.drop(['question'],axis=1,inplace=True)
X_test_tdif.drop(['question'],axis=1,inplace=True)
```

In [0]:

```
# X_test_tdif.head(1)
```

In [48]:

```
print(type(X_train_tdif))
print(type(X_train_tdif.values))
print(X_train_tdif.values)
# print()
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'numpy.ndarray'>
[[0.2857102  0.24999688 0.44443951 ... 0.18181818 2.         0.        ]
 [0.         0.         0.49999167 ... 0.10714286 3.         1.        ]
 [0.74998125 0.74998125 0.99996667 ... 0.42857143 2.         0.        ]
 ...
 [0.599988   0.49999167 0.         ... 0.         2.         0.        ]
 [0.66664445 0.399992   0.499975   ... 0.23076923 6.         4.        ]
 [0.         0.         0.33332778 ... 0.05555556 2.         0.        ]]
```

In [0]:

```
from scipy.sparse import csr_matrix, hstack
Xtr = csr_matrix(X_train_tdif.values)
Xte = csr_matrix(X_test_tdif.values)
```

stacking the data

In [0]:

```
X_tr_final = hstack((Xtr,X_train_question_tfidf)).tocsr()
X_te_final = hstack((Xte,X_test_question_tfidf)).tocsr()
```

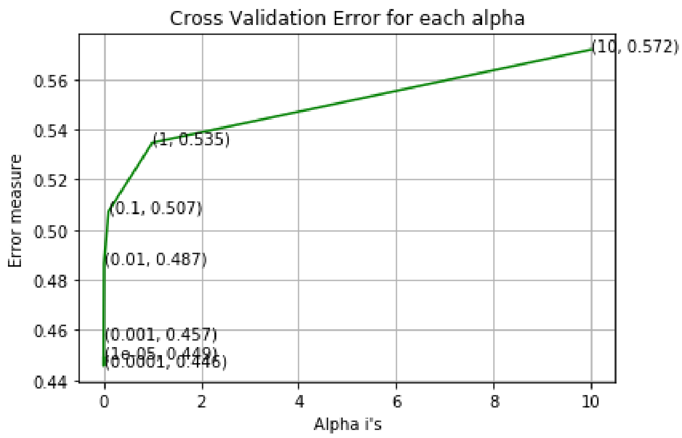
logistic regression

In [51]:

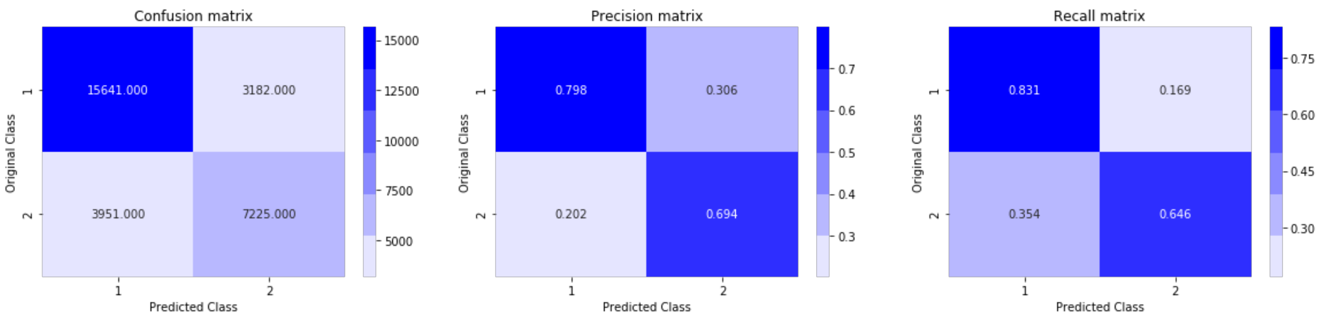
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', class_weight='balanced', random_state=42)
    clf.fit(X_tr_final, y_train_tdif)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr_final, y_train_tdif)
    predict_y = sig_clf.predict_proba(X_te_final)
    log_error_array.append(log_loss(y_test_tdif, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test_tdif, predict_y, labels=clf.classes_, eps=1e-15))

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


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

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

```
predict_y = sig_clf.predict_proba(X_te_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_tdif, 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_tdif, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.4490815726865206
For values of alpha =  0.0001 The log loss is: 0.44567587708209017
For values of alpha =  0.001 The log loss is: 0.45681719091051576
For values of alpha =  0.01 The log loss is: 0.4866837089216423
For values of alpha =  0.1 The log loss is: 0.5072952038448704
For values of alpha =  1 The log loss is: 0.5346835943548855
For values of alpha =  10 The log loss is: 0.5718005542194372
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4476527789934901
For values of best alpha =  0.0001 The test log loss is: 0.44567587708209017
Total number of data points : 29999
```



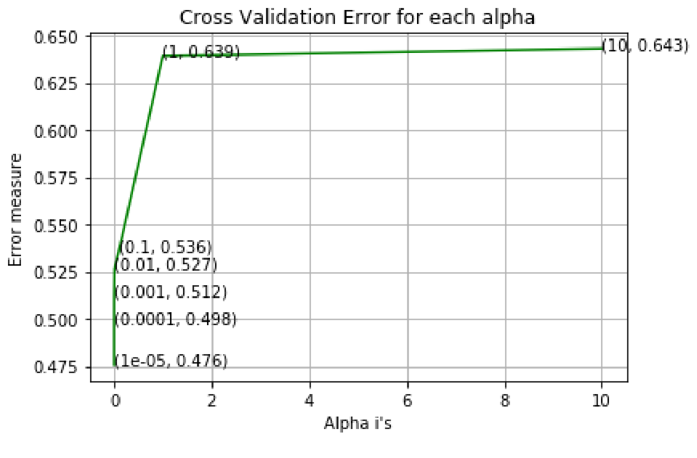
SVM

In [52]:

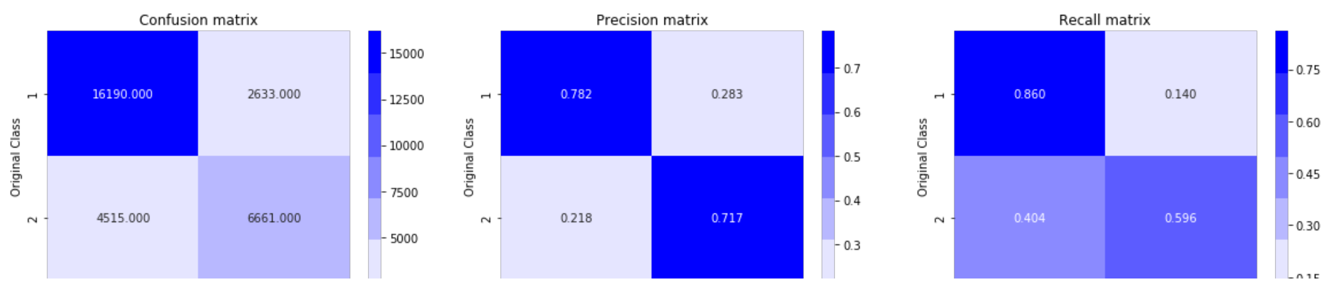
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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:
```

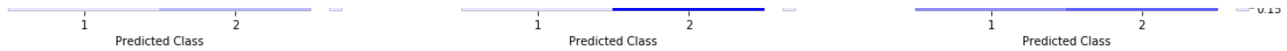
```
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_tr_final, y_train_tdif)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr_final, y_train_tdif)
    predict_y = sig_clf.predict_proba(X_te_final)
    log_error_array.append(log_loss(y_test_tdif, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test_tdif, 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_tr_final, y_train_tdif)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr_final, y_train_tdif)

predict_y = sig_clf.predict_proba(X_tr_final)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_tdif, predict_y, labels=clf.classes_, eps=
1e-15))
predict_y = sig_clf.predict_proba(X_te_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_tdif, predict_y, labels=clf.classes_, eps=1
e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test_tdif, predicted_y)
```

For values of alpha = 1e-05 The log loss is: 0.4756269340349187
For values of alpha = 0.0001 The log loss is: 0.4975739112080212
For values of alpha = 0.001 The log loss is: 0.5120498927141428
For values of alpha = 0.01 The log loss is: 0.5266907873108821
For values of alpha = 0.1 The log loss is: 0.5359308131261725
For values of alpha = 1 The log loss is: 0.6392545439398802
For values of alpha = 10 The log loss is: 0.643123379432156



For values of best alpha = 1e-05 The train log loss is: 0.47345498999409064
For values of best alpha = 1e-05 The test log loss is: 0.4756269340349187
Total number of data points : 29999





XGBOOST

In [0]:

```
#For memory issue batch wise prediction
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

Hyperparameter tunning on Tfidf Weighted W2Vec data

In [54]:

```
# Please write all the code with proper documentation
# Selecting the best alpha using RnadomSearch

#selecting the hyperparameter using RandomSearch
from scipy.stats import randint as sp_randint
import time
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
start_time = time.clock()
xg_clf = XGBClassifier()
parameters = {'n_estimators':sp_randint(5, 1000 ),'max_depth': sp_randint(1, 10)}
clf_xg_1 = RandomizedSearchCV(xg_clf, parameters, cv=3, scoring='neg_log_loss',n_jobs= -1,verbose=10,return_train_score=True
)
clf_xg_1.fit(X_train_final, y_train)

max_depth_list = list(clf_xg_1.cv_results_['param_max_depth'].data)
n_estimator_list = list(clf_xg_1.cv_results_['param_n_estimators'].data)
neg_log_loss = clf_xg_1.cv_results_['mean_test_score']
print("Max Depth = ",max_depth_list)
print("Number of estimators = ",n_estimator_list)
print("Negative log loss = ",neg_log_loss)
for i in range(len(max_depth_list)):
    print("for n_estimators =", n_estimator_list[i],"and max depth = ", max_depth_list[i])
    print("neg log loss= ",neg_log_loss[i] )

best_alpha = np.argmax(neg_log_loss)
print("best log loss= ",neg_log_loss[best_alpha])
print("best n_estimators and max_depth = ",n_estimator_list[best_alpha],'and',max_depth_list[best_alpha])
# print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

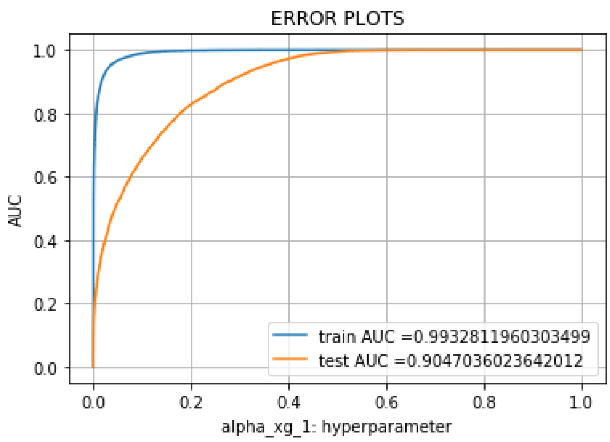
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 69.9min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 133.3min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 177.3min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 226.9min finished
```

```
Max Depth =  [1, 3, 8, 9, 9, 2, 8, 1, 2, 5]
Number of estimators =  [380, 162, 618, 857, 119, 667, 472, 636, 960, 419]
Negative log loss =  [-0.41214218 -0.38656689 -0.39808985 -0.43323535 -0.36838166 -0.38090659
```

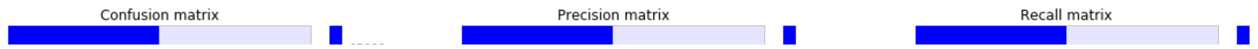
-0.38476648 -0.40503909 -0.37854222 -0.3697411]
for n_estimators = 380 and max depth = 1
neg log loss= -0.4121421777177348
for n_estimators = 162 and max depth = 3
neg log loss= -0.3865668930337774
for n_estimators = 618 and max depth = 8
neg log loss= -0.3980898469159556
for n_estimators = 857 and max depth = 9
neg log loss= -0.43323534852275714
for n_estimators = 119 and max depth = 9
neg log loss= -0.3683816604810309
for n_estimators = 667 and max depth = 2
neg log loss= -0.38090658675171485
for n_estimators = 472 and max depth = 8
neg log loss= -0.3847664757467235
for n_estimators = 636 and max depth = 1
neg log loss= -0.40503908974339614
for n_estimators = 960 and max depth = 2
neg log loss= -0.37854222184091546
for n_estimators = 419 and max depth = 5
neg log loss= -0.3697411000230001
best log loss= -0.3683816604810309
best n_estimators and max_depth = 119 and 9

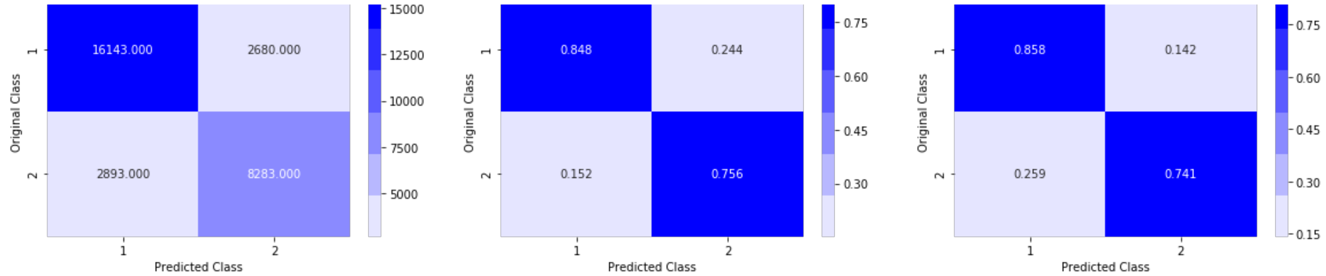
In [57]:

```
n_estimators = 119
max_depth = 9
xg_clf_1 = XGBClassifier(max_depth = max_depth, n_estimators = n_estimators , n_jobs=1)
xg_clf_1.fit(X_train_final, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
y_train_pred_xg_1 = batch_predict(xg_clf_1, X_train_final)
y_test_pred_xg_1 = batch_predict(xg_clf_1, X_test_final)

train_fpr_xg_1, train_tpr_xg_1, tr_thresholds_xg_1 = roc_curve(y_train, y_train_pred_xg_1)
test_fpr_xg_1, test_tpr_xg_1, te_thresholds_xg_1 = roc_curve(y_test, y_test_pred_xg_1)

plt.plot(train_fpr_xg_1, train_tpr_xg_1, label="train AUC ="+str(auc(train_fpr_xg_1, train_tpr_xg_1)))
plt.plot(test_fpr_xg_1, test_tpr_xg_1, label="test AUC ="+str(auc(test_fpr_xg_1, test_tpr_xg_1)))
plt.legend()
plt.xlabel("alpha_xg_1: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

print("Confsion matrix, precision, recall plots")
predicted_y =np.array(np.array(y_test_pred_xg_1)>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```



Confsion matrix, precision, recall plots
Total number of data points : 29999





In [60]:

```
predict_y = xg_clf_1.predict_proba(X_train_final)
# log_loss/
print('For values of best alpha = ', n_estimator_list[best_alpha], 'Max_depth= ',max_depth_list[best_alpha], "The train log loss is:",log
_loss(y_train, predict_y, labels=xg_clf_1.classes_, eps=1e-15))
predict_y = xg_clf_1.predict_proba(X_test_final)
print('For values of best alpha = ', n_estimator_list[best_alpha], 'Max_depth= ',max_depth_list[best_alpha], "The test log loss is:",log_l
oss(y_test, predict_y, labels=xg_clf_1.classes_, eps=1e-15))
# print(predict_y)
# print(type(predict_y))
```

```
For values of best alpha =  119 Max_depth=  9 The train log loss is: 0.1946405829977869
For values of best alpha =  119 Max_depth=  9 The test log loss is: 0.3642663376609388
```

In [0]:

```

```

Conclusions

Step by Step explanation for the solution

1. First I identify which type of machine learning problem this case study is.
2. Then I did the exploratory data analysis such as : Distribution of data points among output classes, Number of unique questions, Number of occurrences of each question etc .
3. Then I removed the null values.
4. Now before processing the text Extracted so basic feature which were as follows
   A. Frequency of qid1's
   B. Frequency of qid2's
   C. Length of q1
   D. Length of q2
   E. Number of words in Question 1
   F. Number of words in Question 2
   G. Number of common unique words in Question 1 and Question 2
   H. word Total in question 1 and question 2
   I. word share between question 1 and question 2
   J. sum total of frequency of qid1 and qid2
   K. absolute difference of frequency of qid1 and qid2
5. Then I did some analysis on these extracted data
6. After that I did the preprocessing of the text data
7. Then I did some advanced feature extraction
8. Then I did some word cloud plot to know some frequent occusing words for both the class 0 and 1
9. Then visualiz the data using T-SNE
10. Then I vectorize the data using tfidf and tfidf weighted w2vec
11. After that I trained a random model to get the clue of the max value of the log-loss
12. Then I trained the logistic regression, linear SVM and XGBOOST models and calculated there precission and recall values for both class 0 and 1

Model comparision

models trained on tfidf weighted w2vec without hyperparameter tunning for XGBOOST

In [62]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["MODEL","VECORIZATION","MIN_LOG_LOSS(test)","PRECISSION Class 1","PRECISSION Class 2","RECALL Cl
ass 1","RECALL Class 2"]
x.add_row(["LOGISTIC REGRESSION", 'weighted w2vec', 0.505,74.0,70.9,88.4,47.7])
x.add_row(["SVM", 'weighted w2vec', 0.527,71.6,74.0,92.0,38.5])
x.add_row(["XGBOOST (without parameter tunning)", 'weighted w2vec', 0.389,83.1,73.7,85.0,70.8])
print(x)
```

```
+-------------------------------------+----------------+--------------------+--------------------+--------------------+----------------+----------------+
|                MODEL                |  VECORIZATION  | MIN_LOG_LOSS(test) | PRECISSION Class 1 | PRECISSION Class 2 | RECALL
Class 1 | RECALL Class 2 |
+-------------------------------------+----------------+--------------------+--------------------+--------------------+----------------+----------------+
|         LOGISTIC REGRESSION         | weighted w2vec |       0.505        |        74.0        |        70.9        |      88.4      |      47.7      |
|                 SVM                 | weighted w2vec |       0.527        |        71.6        |        74.0        |      92.0      |      38.5      |
| XGBOOST (without parameter tunning) | weighted w2vec |       0.389        |        83.1        |        73.7        |      85.0      |      70.8      |
+-------------------------------------+----------------+--------------------+--------------------+--------------------+----------------+----------------+
```

Model trained on simple tfidf with hyperparamter tunning for XGBOOST(trained on tfidf weighted w2vec)

In [63]:

```
y = PrettyTable()
y.field_names = ["MODEL","VECORIZATION","MIN_LOG_LOSS(test)","PRECISSION Class 1","PRECISSION Class 2","RECALL Cl
ass 1","RECALL Class 2"]
y.add_row(["LOGISTIC REGRESSION", 'simple tfidf', 0.455,79.6,69.4,83.1,64.6])
y.add_row(["SVM", 'simple tfidf', 0.475,78.2,71.7,86.0,59.6])
y.add_row(["XGBOOST(with parameter tunning on tfidf weighted w2vec vector)", 'weighted w2vec', 0.364,84.8,75.6,85.8,74.1])
print(y)
```

```
+-----------------------------------------------------------------+----------------+--------------------+--------------------+--------------------+----------------
+----------------+
|                              MODEL                              |  VECORIZATION  | MIN_LOG_LOSS(test) | PRECISSION Class 1 | PRECISSION Cl
ass 2 | RECALL Class 1 | RECALL Class 2 |
+-----------------------------------------------------------------+----------------+--------------------+--------------------+--------------------+----------------
+----------------+
|                       LOGISTIC REGRESSION                       |  simple tfidf  |       0.455        |        79.6        |        69.4        |      83.1      |
64.6      |
|                               SVM                               |  simple tfidf  |       0.475        |        78.2        |        71.7        |      86.0      |      59.6      |
| XGBOOST(with parameter tunning on tfidf weighted w2vec vector) | weighted w2vec |       0.364        |        84.8        |        75.6
|      85.8      |      74.1      |
+-----------------------------------------------------------------+----------------+--------------------+--------------------+--------------------+----------------
+----------------+
```

In [0]: