In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier


from sklearn import preprocessing
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
from tqdm import tqdm
import spacy
import sys
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [7]:

```
"""#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',
t_word_eq','abs_len_diff','mean_len','token_set_ratio','token_sort_ratio','fuzz_ratio','fuzz_partia
tio','longest_substr_ratio','freq_qid1','freq_qid2','q1len','q2len','q1_n_words','q2_n_words','wor
```

```
mon','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',
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_
0_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','4
'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',
','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
_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_
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
3_x','104_x','105_x','106_x','107_x','108_x','109_x','110_x','111_x','112_x','113_x','114_x','115_
16_x','117_x','118_x','119_x','120_x','121_x','122_x','123_x','124_x','125_x','126_x','127_x','128_
129_x','130_x','131_x','132_x','133_x','134_x','135_x','136_x','137_x','138_x','139_x','140_x','14
'142_x','143_x','144_x','145_x','146_x','147_x','148_x','149_x','150_x','151_x','152_x','153_x','1
,'155_x','156_x','157_x','158_x','159_x','160_x','161_x','162_x','163_x','164_x','165_x','166_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',
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',
_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
6_x','207_x','208_x','209_x','210_x','211_x','212_x','213_x','214_x','215_x','216_x','217_x','218_
19_x','220_x','221_x','222_x','223_x','224_x','225_x','226_x','227_x','228_x','229_x','230_x','231_
232_x','233_x','234_x','235_x','236_x','237_x','238_x','239_x','240_x','241_x','242_x','243_x','24
'245_x','246_x','247_x','248_x','249_x','250_x','251_x','252_x','253_x','254_x','255_x','256_x','2
,'258_x','259_x','260_x','261_x','262_x','263_x','264_x','265_x','266_x','267_x','268_x','269_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',
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',
_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
9_x','310_x','311_x','312_x','313_x','314_x','315_x','316_x','317_x','318_x','319_x','320_x','321_
22_x','323_x','324_x','325_x','326_x','327_x','328_x','329_x','330_x','331_x','332_x','333_x','334_
335_x','336_x','337_x','338_x','339_x','340_x','341_x','342_x','343_x','344_x','345_x','346_x','34
'348_x','349_x','350_x','351_x','352_x','353_x','354_x','355_x','356_x','357_x','358_x','359_x','3
,'361_x','362_x','363_x','364_x','365_x','366_x','367_x','368_x','369_x','370_x','371_x','372_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
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_
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','
,'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',
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_
3_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','7
'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',
','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',
_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
9_y','120_y','121_y','122_y','123_y','124_y','125_y','126_y','127_y','128_y','129_y','130_y','131_
32_y','133_y','134_y','135_y','136_y','137_y','138_y','139_y','140_y','141_y','142_y','143_y','144_
145_y','146_y','147_y','148_y','149_y','150_y','151_y','152_y','153_y','154_y','155_y','156_y','15
'158_y','159_y','160_y','161_y','162_y','163_y','164_y','165_y','166_y','167_y','168_y','169_y','1
,'171_y','172_y','173_y','174_y','175_y','176_y','177_y','178_y','179_y','180_y','181_y','182_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',
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',
_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
2_y','223_y','224_y','225_y','226_y','227_y','228_y','229_y','230_y','231_y','232_y','233_y','234_
35_y','236_y','237_y','238_y','239_y','240_y','241_y','242_y','243_y','244_y','245_y','246_y','247_
248_y','249_y','250_y','251_y','252_y','253_y','254_y','255_y','256_y','257_y','258_y','259_y','26
'261_y','262_y','263_y','264_y','265_y','266_y','267_y','268_y','269_y','270_y','271_y','272_y','2
,'274_y','275_y','276_y','277_y','278_y','279_y','280_y','281_y','282_y','283_y','284_y','285_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',
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',
_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
5_y','326_y','327_y','328_y','329_y','330_y','331_y','332_y','333_y','334_y','335_y','336_y','337_
38_y','339_y','340_y','341_y','342_y','343_y','344_y','345_y','346_y','347_y','348_y','349_y','350_
351_y','352_y','353_y','354_y','355_y','356_y','357_y','358_y','359_y','360_y','361_y','362_y','36
'364_y','365_y','366_y','367_y','368_y','369_y','370_y','371_y','372_y','373_y','374_y','375_y','3
,'377_y','378_y','379_y','380_y','381_y','382_y','383_y'], chunksize=chunksize, iterator=True, enc
oding='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 [8]:

```
#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:
```

```
    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 [9]:

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

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

In [10]:

```
"""# 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 50000;", conn_r)
        conn_r.commit()
        conn_r.close()"""
```

In [11]:

```
"""# 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)"""
```

4.2 Converting strings to numerics

In [2]:

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

Out[2]:

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

In [15]:

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

In [2]:

```python
# avoid decoding problems
new_df = pd.read_csv("train_new.csv")

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
# ----------------- python 3 ---------------------
new_df['question1'] = new_df['question1'].apply(lambda x: str(x))
new_df['question2'] = new_df['question2'].apply(lambda x: str(x))
new_df.head()
```

Out[2]:

	Unnamed: 0	id	qid1	qid2	question1	question2	is_duplicate
0	237030	237030	33086	348102	How can I stop playing video games?	Should I stop playing video games with my child?	0
1	247341	247341	73272	8624	Who is better Donald Trump or Hillary Clinton?	Why is Hillary Clinton a better choice than Do...	1
2	246425	246425	359482	359483	What do you think is the chance that sometime ...	Do you think there will be another world war/n...	1
3	306985	306985	1357	47020	Why are so many questions posted to Quora that...	Why do people write questions on Quora that co...	1
4	225863	225863	334315	334316	Can there even be a movie ever rated 10/10 on ...	What are your 10/10 movies?	0

Building Features and Splitting the Data

In [3]:

```python
df_basic_feature = pd.read_csv("df_fe_without_preprocessing_train_50k.csv",encoding='latin-1')
```

In [4]:

```python
df_basic_feature.columns
```

Out[4]:

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

In [5]:

```python
df_basic_feature.head()
```

Out[5]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Co
0	237030	33086	348102	How can I stop playing video games?	Should I stop playing video games with my child?	0	1	1	35	48	7	9	
1	247341	73272	8624	Who is better Donald Trump or Hillary	Why is Hillary Clinton a better choice	1	3	3	46	57	8	10	

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Co
				Clinton?	than Do...								
2	246425	359482	359483	What do you think is the chance that sometime ...	Do you think there will be another world war/n...	1	1	1	139	77	28	14	
3	306985	1357	47020	Why are so many questions posted to Quora that...	Why do people write questions on Quora that co...	1	4	4	86	86	16	16	
4	225863	334315	334316	Can there even be a movie ever rated 10/10 on ...	What are your 10/10 movies?	0	1	1	51	27	11	5	

In [6]:

```python
df_advance_features = pd.read_csv("nlp_features_train_50k.csv",encoding='latin-1')
```

In [7]:

```python
df_advance_features.columns
```

Out[7]:

```
Index(['Unnamed: 0', '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')
```

In [8]:

```python
df_advance_features.head()
```

Out[8]:

	Unnamed: 0	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	...	ctc_max	last_word_eq
0	237030	237030	33086	348102	how can i stop playing video games	should i stop playing video games with my child	0	0.999975	0.799984	0.333322	...	0.555549	0.0
1	247341	247341	73272	8624	who is better donald trump or hillary clinton	why is hillary clinton a better choice than do...	1	0.999980	0.833319	0.333322	...	0.599994	0.0
2	246425	246425	359482	359483	what do you think is the chance that sometime ...	do you think there will be another world war n...	1	0.857131	0.499996	0.999986	...	0.464284	0.0
3	306985	306985	1357	47020	why are so many questions posted to quora that...	why do people write questions on quora that co...	1	0.374995	0.333330	0.333328	...	0.312498	0.0
4	225863	225863	334315	334316	can there even be a movie ever rated	what are your 10 10	0	0.499975	0.166664	0.000000	...	0.083333	0.0

Unnamed: 0	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	...	ctc_max	last_word_eq

5 rows × 22 columns

In [9]:

```
# Columns dropped from basic feature dataframe
df1 = df_basic_feature.drop(['qid1','qid2','question1','question2'],axis=1)

# Columns dropped from advance feature dataframe
df2 = df_advance_features.drop(['Unnamed: 0','qid1','qid2','question1','question2','is_duplicate'],
axis=1)
```

In [10]:

```
df3 = df_advance_features[['id','question1','question2']]
```

In [11]:

```
print(df1.columns)
print(df2.columns)
print(df3.columns)
```

```
Index(['id', 'is_duplicate', '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')
Index(['id', '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', 'question1', 'question2'], dtype='object')
```

In [12]:

```
df3 = df3.fillna(' ')
new_df_q = pd.DataFrame()
new_df_q['questions'] = df3.question1 + ' ' + df3.question2
new_df_q['id'] = df3.id
df2['id']=df1['id']
new_df_q['id']=df1['id']
final_df = df1.merge(df2, on='id',how='left') #merging df1 and df2
X  = final_df.merge(new_df_q, on='id',how='left')#merging final_df and new_df
```

In [13]:

```
#removing id from X
X=X.drop('id',axis=1)
X.columns
```

Out[13]:

```
Index(['is_duplicate', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len',
       'q1_n_words', 'q2_n_words', 'word_Common', 'word_Total', 'word_share',
       'freq_q1+q2', 'freq_q1-q2', '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', 'questions'],
      dtype='object')
```

In [14]:

```
print(len(X.columns))
```

```
28
```

Checking for Null values

In [15]:

```
nan_rows = X[X.isnull().any(1)]
print(nan_rows)
```

```
Empty DataFrame
Columns: [is_duplicate, freq_qid1, freq_qid2, q1len, q2len, q1_n_words, q2_n_words, word_Common, w
ord_Total, word_share, freq_q1+q2, freq_q1-q2, cwc_min, cwc_max, csc_min, csc_max, ctc_min, ctc_ma
x, 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, questions]
Index: []

[0 rows x 28 columns]
```

In [16]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 28 columns):
is_duplicate            50000 non-null int64
freq_qid1               50000 non-null int64
freq_qid2               50000 non-null int64
q1len                   50000 non-null int64
q2len                   50000 non-null int64
q1_n_words              50000 non-null int64
q2_n_words              50000 non-null int64
word_Common             50000 non-null float64
word_Total              50000 non-null float64
word_share              50000 non-null float64
freq_q1+q2              50000 non-null int64
freq_q1-q2              50000 non-null int64
cwc_min                 50000 non-null float64
cwc_max                 50000 non-null float64
csc_min                 50000 non-null float64
csc_max                 50000 non-null float64
ctc_min                 50000 non-null float64
ctc_max                 50000 non-null float64
last_word_eq            50000 non-null float64
first_word_eq           50000 non-null float64
abs_len_diff            50000 non-null float64
mean_len                50000 non-null float64
token_set_ratio         50000 non-null int64
token_sort_ratio        50000 non-null int64
fuzz_ratio              50000 non-null int64
fuzz_partial_ratio      50000 non-null int64
longest_substr_ratio    50000 non-null float64
questions               50000 non-null object
dtypes: float64(14), int64(13), object(1)
memory usage: 11.1+ MB
```

In [17]:

```
# Seperate the is_duplicate feature
y = X['is_duplicate']
```

In [18]:

```
#Drop id and is_duplicate features
X.drop(['is_duplicate'], axis=1, inplace=True)
```

In [19]:

```
print(X.columns)
print("Total no. of features : " , len(X.columns))
```

```
Index(['freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
```

```
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2',
       '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', 'questions'],
      dtype='object')
Total no. of features :  27
```

4.3 Random train test split(70:30)

In [31]:

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

In [32]:

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

```
(35000, 27)
(35000,)
(15000, 27)
(15000,)
```

In [33]:

```
#seperating questions for tfidf vectorizer
X_train_q=X_train['questions']
X_test_q=X_test['questions']

X_train=X_train.drop('questions',axis=1)
X_test=X_test.drop('questions',axis=1)
```

Standardize Data

In [34]:

```
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

TFIDF Weighted W2V

In [35]:

```
tfidf1 = TfidfVectorizer(lowercase=False, )
tfidf1.fit_transform(X_train_q)

# dict key:word and value:tf-idf score
word2tfidf1 = dict(zip(tfidf1.get_feature_names(), tfidf1.idf_))
```

1. After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores.
2. here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usage/vectors-similarity
3. It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [36]:

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
#for train dataset

nlp = spacy.load('en')
```

```python
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_train_q)):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf1[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
#df['q1_feats_m'] = list(vecs1)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 35000/35000 [05:
07<00:00, 113.67it/s]
```

In [37]:

```python
vecs2 = []
for qu2 in tqdm(list(X_test_q)):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 96])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf1[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)
#df['q2_feats_m'] = list(vecs2)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 15000/15000 [02:
19<00:00, 107.84it/s]
```

In [38]:

```python
first_df=pd.DataFrame(vecs1)
sec_df=pd.DataFrame(vecs2)
```

In [39]:

```python
from scipy.sparse import hstack
X_train = hstack((X_train,first_df))
X_test= hstack((X_test,sec_df))
print(X_train.shape)
print(X_test.shape)
```

```
(35000, 122)
(15000, 122)
```

TFIDF

In [40]:

```python
#splitting data into train and test
x_train,x_test,y_train,y_test=train_test_split(X,y,random_state=3,test_size=0.3)
```

In [41]:

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(35000, 27)
(35000,)
(15000, 27)
(15000,)
```

In [42]:

```
#seperating questions for tfidf vectorizer
x_train_q=x_train['questions']
x_test_q=x_test['questions']

x_train=x_train.drop('questions',axis=1)
x_test=x_test.drop('questions',axis=1)
```

In [43]:

```
#tfidf vectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,3),min_df=10)
x_train_tfidf=tf_idf_vect.fit_transform(x_train_q)
x_test_tfidf=tf_idf_vect.transform(x_test_q)
```

In [44]:

```
#adding tfidf features to our train and test data using hstack
x_train = hstack((x_train,x_train_tfidf))
x_test= hstack((x_test.values,x_test_tfidf))
print(x_train.shape)
print(x_test.shape)
```

```
(35000, 14522)
(15000, 14522)
```

==

In [15]:

```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len)
print("-"*10, "Distribution of output variable in 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:  0.6316180462298923 Class 1:  0.36838195377010774
---------- Distribution of output variable in train data ----------
Class 0:  0.3684 Class 1:  0.3684
```

In [45]:

```
# 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
```

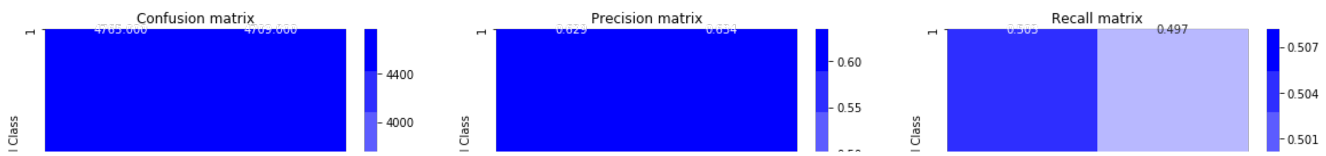
```
    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #         [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [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()
```

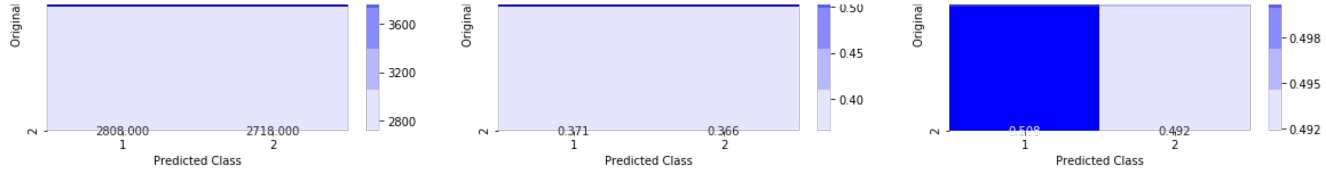
4.4 Building a random model (Finding worst-case log-loss)

In [17]:

```
# 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 0.8870507979822069
```





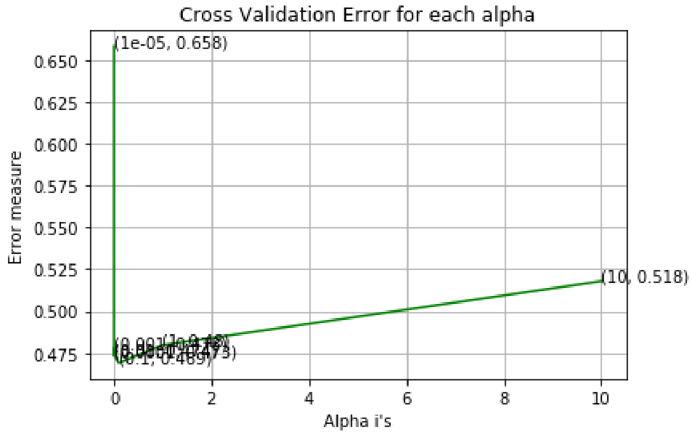
4.4 Logistic Regression with hyperparameter tuning

In [18]:

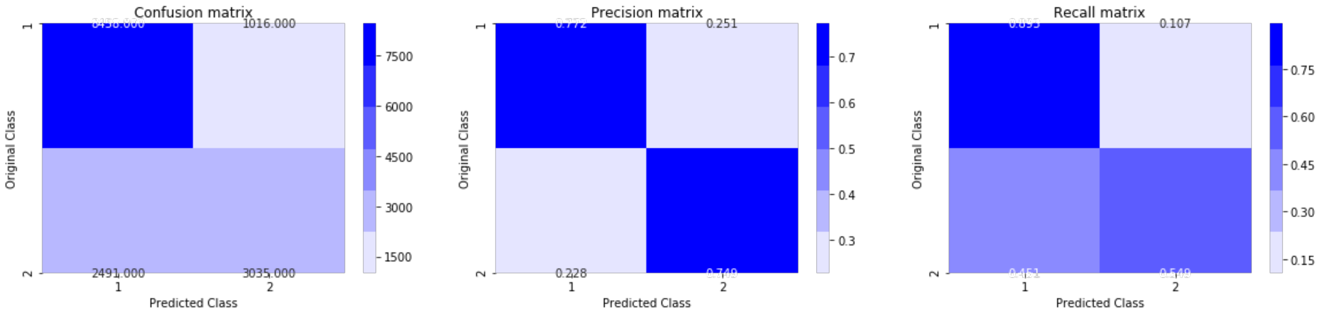
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 0.6580986393397351
For values of alpha =  0.0001 The log loss is: 0.4734825006756204
For values of alpha =  0.001 The log loss is: 0.47759230096251687
For values of alpha =  0.01 The log loss is: 0.4736042222288211
```

```
For values of alpha =  0.01 The log loss is: 0.4750042222206211
For values of alpha =  0.1 The log loss is: 0.4690289001423947
For values of alpha =  1 The log loss is: 0.4796846761677882
For values of alpha =  10 The log loss is: 0.5178282170965716
```




```
For values of best alpha =  0.1 The train log loss is: 0.45435664933273684
For values of best alpha =  0.1 The test log loss is: 0.4690289001423947
Total number of data points : 15000
```



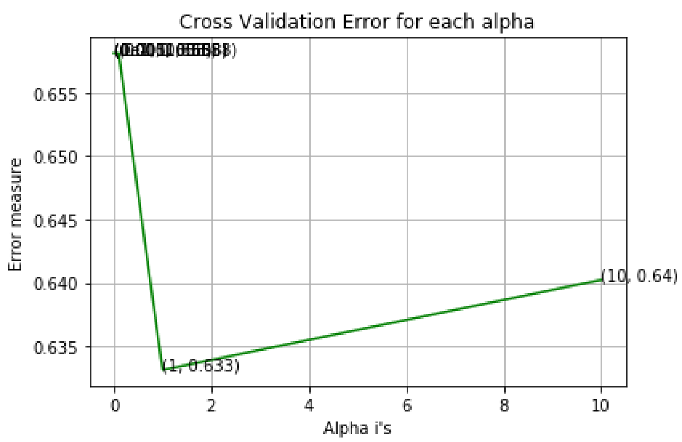
4.5 Linear SVM with hyperparameter tuning

In [19]:

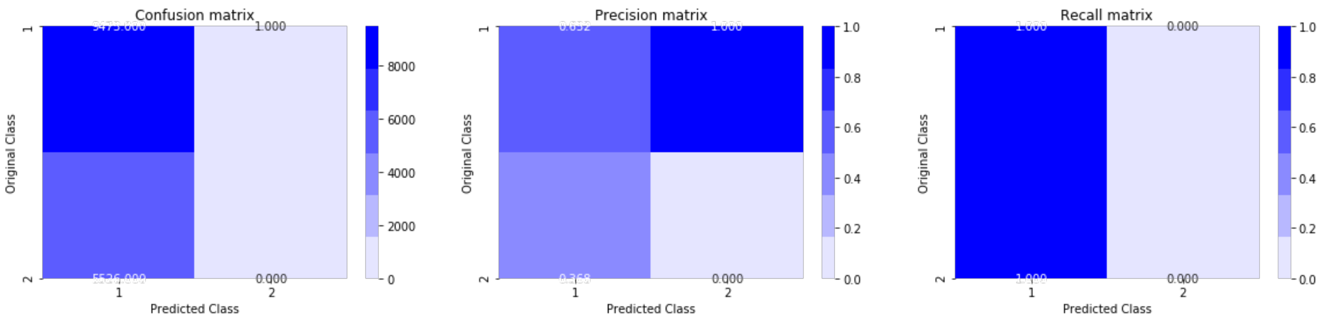
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/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: 0.6580986393397351
For values of alpha =  0.0001 The log loss is: 0.6580986393397351
For values of alpha =  0.001 The log loss is: 0.6580986393397351
For values of alpha =  0.01 The log loss is: 0.6580986393397351
For values of alpha =  0.1 The log loss is: 0.6580986393397351
For values of alpha =  1 The log loss is: 0.6331291263177062
For values of alpha =  10 The log loss is: 0.6402243040223459
```




```
For values of best alpha =  1 The train log loss is: 0.6335520279730955
For values of best alpha =  1 The test log loss is: 0.6331291263177062
Total number of data points : 15000
```



4.6 XGBoost

In [20]:

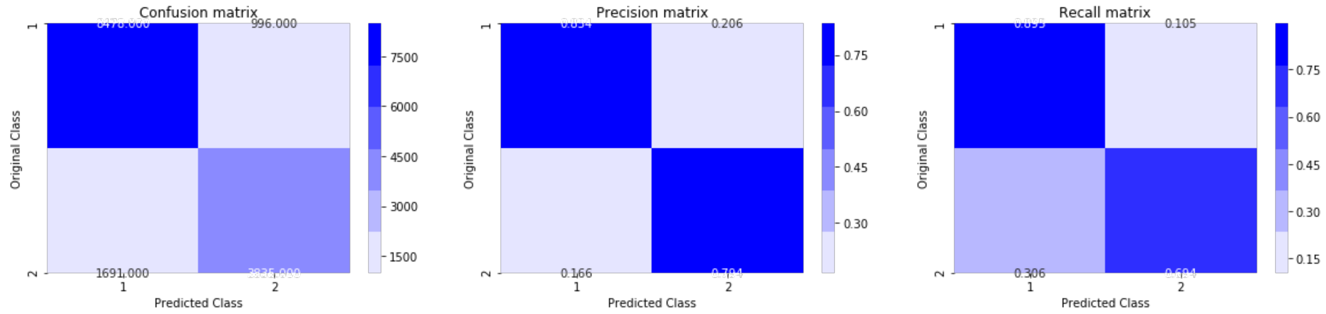
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

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

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

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

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

```
[0] train-logloss:0.68468 valid-logloss:0.684777
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:0.614771 valid-logloss:0.615316
[20] train-logloss:0.564011 valid-logloss:0.564956
[30] train-logloss:0.525906 valid-logloss:0.527275
[40] train-logloss:0.496649 valid-logloss:0.498375
[50] train-logloss:0.473932 valid-logloss:0.476049
[60] train-logloss:0.455721 valid-logloss:0.458301
[70] train-logloss:0.440928 valid-logloss:0.443979
[80] train-logloss:0.428998 valid-logloss:0.432604
[90] train-logloss:0.419384 valid-logloss:0.423584
[100] train-logloss:0.411192 valid-logloss:0.41595
[110] train-logloss:0.40438 valid-logloss:0.40968
[120] train-logloss:0.398461 valid-logloss:0.404326
[130] train-logloss:0.393695 valid-logloss:0.400196
[140] train-logloss:0.389263 valid-logloss:0.396366
[150] train-logloss:0.385428 valid-logloss:0.393076
[160] train-logloss:0.382122 valid-logloss:0.390319
[170] train-logloss:0.379009 valid-logloss:0.387803
[180] train-logloss:0.376375 valid-logloss:0.3857
[190] train-logloss:0.373958 valid-logloss:0.383855
[200] train-logloss:0.371759 valid-logloss:0.382191
[210] train-logloss:0.36975 valid-logloss:0.380682
[220] train-logloss:0.367737 valid-logloss:0.379227
[230] train-logloss:0.365769 valid-logloss:0.377795
[240] train-logloss:0.363911 valid-logloss:0.376501
[250] train-logloss:0.36197 valid-logloss:0.375205
[260] train-logloss:0.359843 valid-logloss:0.373705
[270] train-logloss:0.357923 valid-logloss:0.372473
[280] train-logloss:0.356201 valid-logloss:0.371329
[290] train-logloss:0.354439 valid-logloss:0.370231
[300] train-logloss:0.352729 valid-logloss:0.369173
[310] train-logloss:0.351111 valid-logloss:0.368279
[320] train-logloss:0.349599 valid-logloss:0.367549
[330] train-logloss:0.347964 valid-logloss:0.366605
[340] train-logloss:0.346428 valid-logloss:0.365835
[350] train-logloss:0.344863 valid-logloss:0.365089
[360] train-logloss:0.343448 valid-logloss:0.364412
[370] train-logloss:0.342006 valid-logloss:0.363753
[380] train-logloss:0.340605 valid-logloss:0.363019
[390] train-logloss:0.33917 valid-logloss:0.362462
[399] train-logloss:0.338052 valid-logloss:0.361986
The test log loss is: 0.36198559796476426
```

In [21]:

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

```
Total number of data points : 15000
```



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.

Apply ML Models

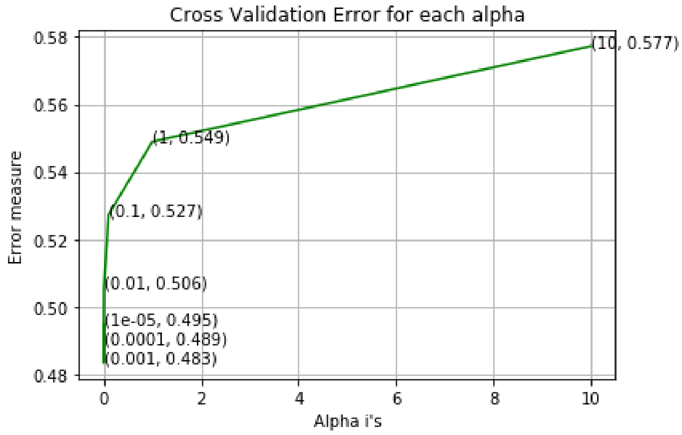
1. Logistic Regression with hyperparameter tuning

In [47]:

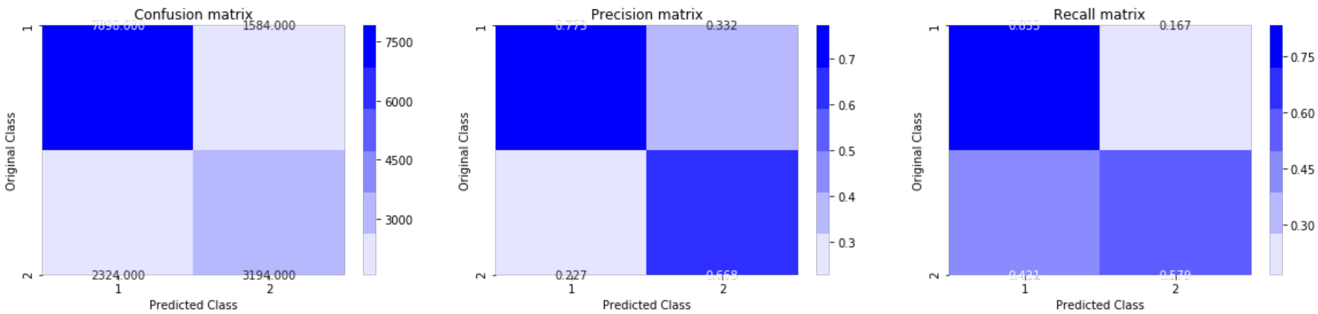
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, class_weight='balanced'
)
    clf.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    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, class_weigh
t='balanced')
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: 0.4947422210879909
For values of alpha =  0.0001 The log loss is: 0.489154151853496
For values of alpha =  0.001 The log loss is: 0.4833711726613907
For values of alpha =  0.01 The log loss is: 0.5056967278617175
```

```
For values of alpha =  0.1 The log loss is: 0.5272471131570954
For values of alpha =  1 The log loss is: 0.5489549273504452
For values of alpha =  10 The log loss is: 0.5772296942076715
```




```
For values of best alpha =  0.001 The train log loss is: 0.477906100156144
For values of best alpha =  0.001 The test log loss is: 0.4833711726613907
Total number of data points : 15000
```



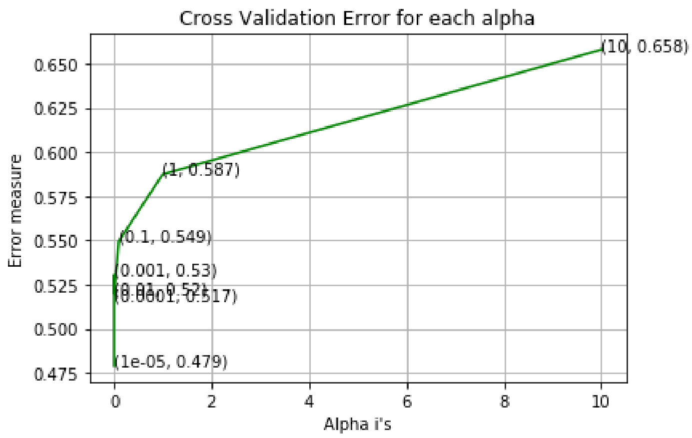
2. Linear SVM with hyperparameter tuning

In [48]:

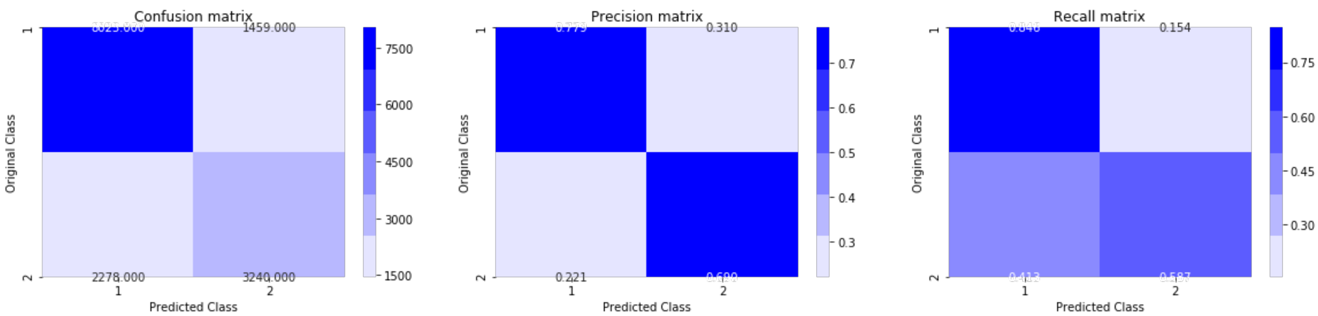
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42, class_weight='balance
d')
    clf.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    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,
class_weight='balanced')
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: 0.4788677700048385
For values of alpha =  0.0001 The log loss is: 0.5165486747686439
For values of alpha =  0.001 The log loss is: 0.5303941928647448
For values of alpha =  0.01 The log loss is: 0.5197592602603484
For values of alpha =  0.1 The log loss is: 0.5493696876820111
For values of alpha =  1 The log loss is: 0.5873010378420138
For values of alpha =  10 The log loss is: 0.6578236219028443
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4728582429930364
For values of best alpha =  1e-05 The test log loss is: 0.4788677700048385
Total number of data points : 15000
```



3. XGBoost with Hyperparameter tuning

In [40]:

```
from xgboost import XGBClassifier
parameters = {"learning_rate":[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4], "max_depth":[3, 4, 6, 7, 9,
11, 15]}

clf = XGBClassifier()
clf_ran = RandomizedSearchCV(clf, parameters, cv = 3)
ran_fit = clf_ran.fit(X_train, y_train)

best_learning_rate = ran_fit.best_estimator_.learning_rate
best_max_depth = ran_fit.best_estimator_.max_depth

print("_"*80)
print("After Hyperparameter tuning we get\n")
print("Best Learning Rate : ", best_learning_rate)
print("Best Max Depth : ", best_max_depth)
```

```
________________________________________________________________________________
After Hyperparameter tuning we get

Best Learning Rate :  0.2
Best Max Depth :  15
```

In [41]:

```
from xgboost import XGBClassifier
parameters = {"n_estimators":[80, 100, 150, 200, 250, 300, 450], "min_child_weight":[1,2, 3, 4, 5, 6
, 7]}

clf = XGBClassifier()
clf_ran = RandomizedSearchCV(clf, parameters, cv = 3)
ran_fit = clf_ran.fit(X_train, y_train)

best_min_child_weight = ran_fit.best_estimator_.min_child_weight
best_n_estimators = ran_fit.best_estimator_.n_estimators

print("_"*80)
print("After Hyperparameter tuning we get\n")
print("Best Min Child Weight : ", best_min_child_weight)
print("Best N_estimator : ", best_n_estimators)
```

```
________________________________________________________________________________
After Hyperparameter tuning we get

Best Min Child Weight :  3
Best N_estimator :  450
```

In [43]:

```
XGB = XGBClassifier(max_depth=15,
                              learning_rate=0.2,
                              n_estimators=450,
                              min_child_weight=3)
XGB
```

Out[43]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.2, max_delta_step=0, max_depth=15,
              min_child_weight=3, missing=None, n_estimators=450, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [47]:

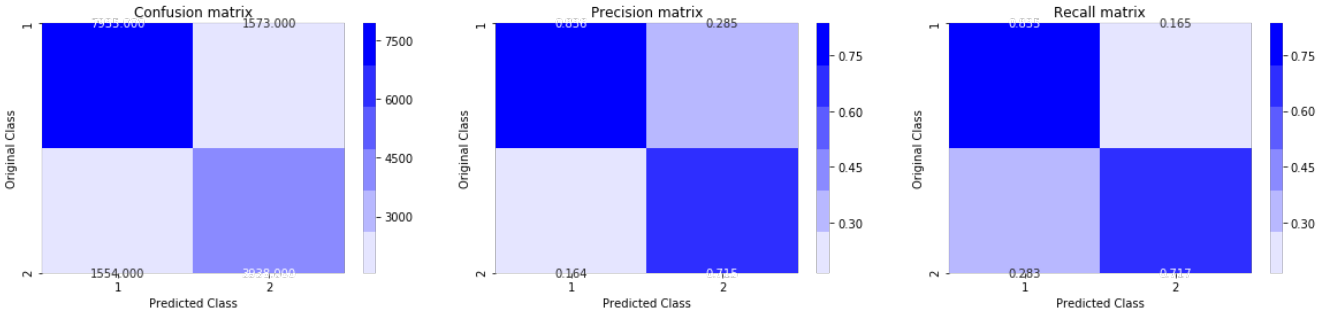
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 15
params['learning_rate'] = 0.2
params['n_estimators'] = 450
params['min_child_weight'] = 3
params['subsample'] = 1
params['gamma'] = 0
params['colsample_bylevel'] = 1

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

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

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

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

```
[0] train-logloss:0.585516 valid-logloss:0.614969
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:0.222701 valid-logloss:0.424452
[20] train-logloss:0.14142 valid-logloss:0.40441
[30] train-logloss:0.103308 valid-logloss:0.402445
[40] train-logloss:0.076166 valid-logloss:0.404883
Stopping. Best iteration:
[27] train-logloss:0.116261 valid-logloss:0.40163
```

In [48]:

```
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
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)
```

```
The test log loss is: 0.40758969216355423
Total number of data points : 15000
```



In [49]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ['Dataset Size', 'Model Name', 'Tokenizer','Hyperparameter Tunning', 'Test Log Loss']
ptable.add_row(["~ 50K","Logistic Regression","TFIDF"," YES","0.48"])
ptable.add_row(["~ 50K","Linear SVM","TFIDF"," YES","0.47"])
ptable.add_row(["~ 50K","XGBoost","TFIDF Weighted W2V"," YES","0.40"])
print(ptable)
```

```
+--------------+---------------------+--------------------+------------------------+---------------
| Dataset Size |      Model Name     |      Tokenizer     | Hyperparameter Tunning | Test Log Loss
|
+--------------+---------------------+--------------------+------------------------+---------------
|    ~ 50K     | Logistic Regression |        TFIDF       |          YES           |      0.48
|    ~ 50K     |      Linear SVM     |        TFIDF       |          YES           |      0.47
|    ~ 50K     |       XGBoost       | TFIDF Weighted W2V |          YES           |      0.40
+--------------+---------------------+--------------------+------------------------+---------------
```

Conclusions

1. Dataset size for models is 50k datapoints.
2. Logistic Regression and Linear SVM works well with high dimensional data.
3. XGBoost model with Hyperparameter tuning has Log Loss of 0.40
4. Basic and Advanced features are combined together to perform TFIDF Vectorization.
5. Standardized our data after splitting in 70:30 ratio.
6. Vectorization is done after splitting the data, X_train and X_test are for TFIDF Weighted W2V and x_train and x_test are for TFIDF vectorization.

In [ ]:

In [ ]: