Quora Question Pairs

1. Business Problem

1.1 Description

Quora is a place to gain and share knowledge—about anything. It’s a platform to ask questions and connect with people who contribute unique insights and quality answers. This empowers people to learn from each other and to better understand the world.

Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

> Credits: Kaggle

Problem Statement

- Identify which questions asked on Quora are duplicates of questions that have already been asked.
- This could be useful to instantly provide answers to questions that have already been answered.
- We are tasked with predicting whether a pair of questions are duplicates or not.

1.2 Sources/Useful Links

- Source : https://www.kaggle.com/c/quora-question-pairs

  Useful Links
- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

- Data will be in a file Train.csv
- Train.csv contains 5 columns : qid1, qid2, question1, question2, is_duplicate
- Size of Train.csv - 60MB
- Number of rows in Train.csv = 404,290

2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in india?","What is
the step by step guide to invest in share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamond?","What would happen if the
Indian government stole the Kohinoor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a great
geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my Youtube
comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.

2.2.2 Performance Metric

Source: https://www.kaggle.com/c/quora-question-pairs#evaluation

Metric(s):

- log-loss : https://www.kaggle.com/wiki/LogarithmicLoss
- Binary Confusion Matrix

2.3 Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

3. Exploratory Data Analysis

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
```

3.1 Reading data and basic stats

In [2]:

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

print("Number of data points:",df.shape[0])
```

```
Number of data points: 404290
```

In [3]:

```
df.head()
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [4]:

```
df.info()
```

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

We are given a minimal number of data fields here, consisting of:

- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

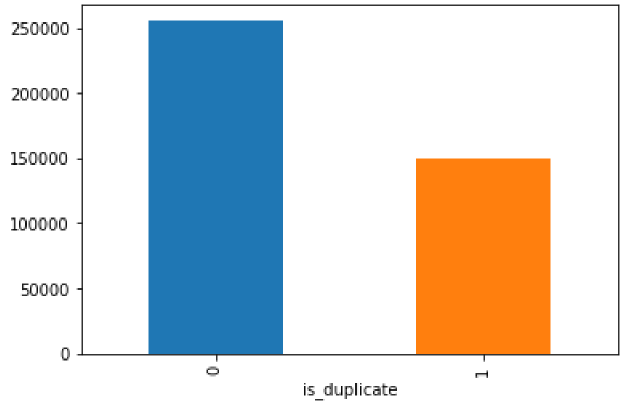
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [5]:

```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15357dd8>
```



In [6]:

```
print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

```
~> Total number of question pairs for training:
   404290
```

In [7]:

```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_duplicate'].mean()*100, 2)))
```

```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

~> Question pairs are Similar (is_duplicate = 1):
   36.92%
```

3.2.2 Number of unique questions

In [8]:

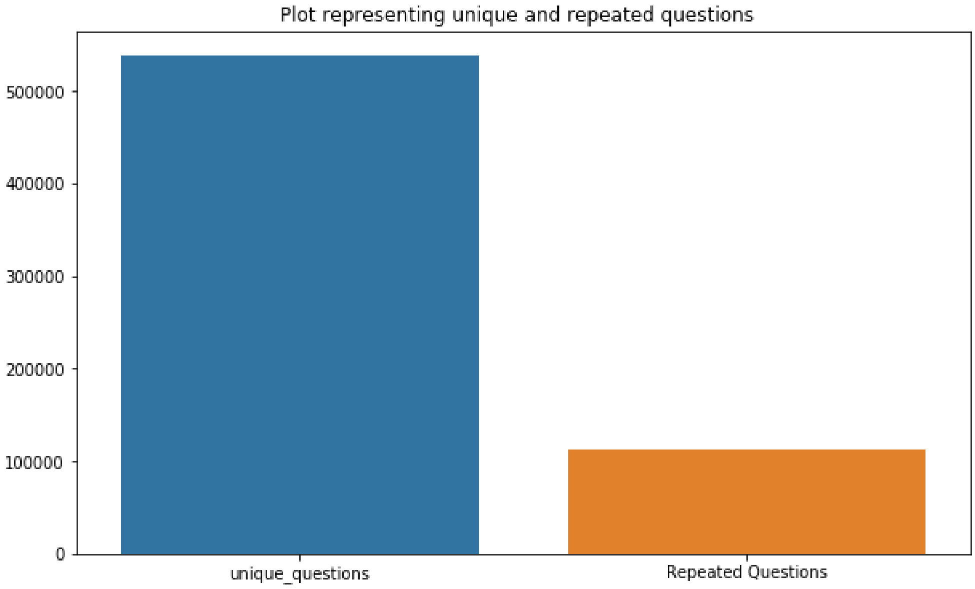
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(qs_morethan_onetime,qs_morethan_onetime/unique_qs*100))

print ('Max number of times a single question is repeated: {}\n'.format(max(qids.value_counts())))


q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.77953945937505%)

Max number of times a single question is repeated: 157
```

In [9]:

```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]
```

```
plt.figure(figsize=(10, 6))
plt.title ("Plot representing unique and repeated questions  ")
sns.barplot(x,y)
plt.show()
```



3.2.3 Checking for Duplicates

In [10]:

```
#checking whether there are any repeated pair of questions

pair_duplicates =
df[['qid1','qid2','is_duplicate']].groupby(['qid1','qid2']).count().reset_index()

print ("Number of duplicate questions",(pair_duplicates).shape[0] - df.shape[0])
```

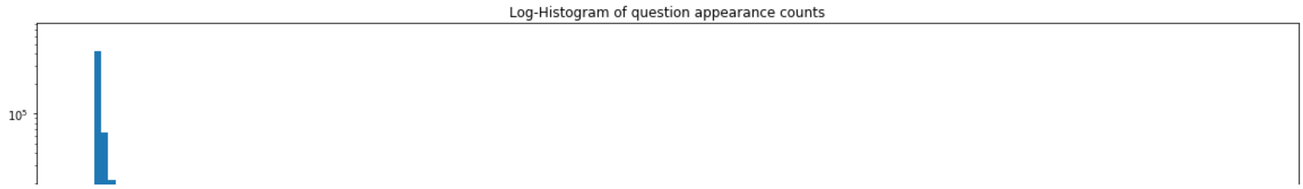
```
Number of duplicate questions 0
```

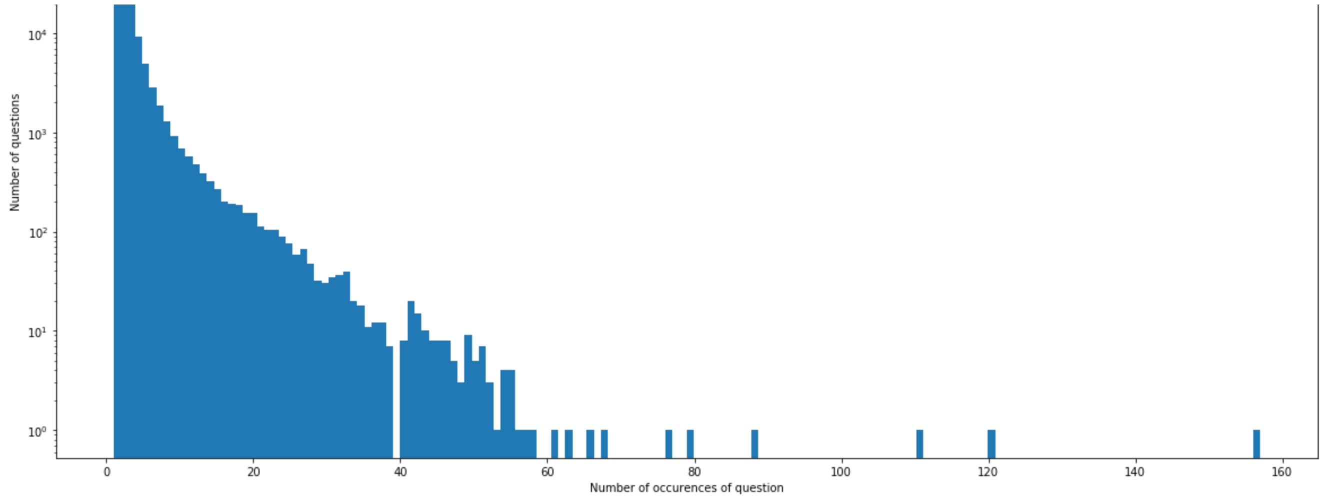
3.2.4 Number of occurrences of each question

In [11]:

```
plt.figure(figsize=(20, 10))

plt.hist(qids.value_counts(), bins=160)

plt.yscale('log', nonposy='clip')

plt.title('Log-Histogram of question appearance counts')

plt.xlabel('Number of occurences of question')

plt.ylabel('Number of questions')

print ('Maximum number of times a single question is repeated: {}\n'.format(max(qids.value_counts(
))))
```

```
Maximum number of times a single question is repeated: 157
```





3.2.5 Checking for NULL values

In [12]:

```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

```
            id    qid1    qid2                         question1  \
105780  105780  174363  174364    How can I develop android app?
201841  201841  303951  174364  How can I create an Android app?
363362  363362  493340  493341                               NaN

                                               question2  is_duplicate
105780                                               NaN             0
201841                                               NaN             0
363362  My Chinese name is Haichao Yu. What English na...             0
```

- There are two rows with null values in question2

In [13]:

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

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
```

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1
- **q2len** = Length of q2
- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2
- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)
- **word_share** = (word_common)/(word_Total)
- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2
- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

In [14]:

```python
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

    def normalized_word_Common(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * len(w1 & w2)
    df['word_Common'] = df.apply(normalized_word_Common, axis=1)

    def normalized_word_Total(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * (len(w1) + len(w2))
    df['word_Total'] = df.apply(normalized_word_Total, axis=1)

    def normalized_word_share(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * len(w1 & w2)/(len(w1) + len(w2))
    df['word_share'] = df.apply(normalized_word_share, axis=1)

    df['freq_q1+q2'] = df['freq_qid1']+df['freq_qid2']
    df['freq_q1-q2'] = abs(df['freq_qid1']-df['freq_qid2'])

    df.to_csv("df_fe_without_preprocessing_train.csv", index=False)

df.head()
```

Out[14]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14	12	10.0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	8	13	4.0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	14	10	4.0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65	11	9	0.0
				Which one dissolve in	Which fish								

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_
4	4	9	10	water quikly sugar, salt...	would survive in salt water?	0	3	1	76	39	13	7	2.0

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [15]:

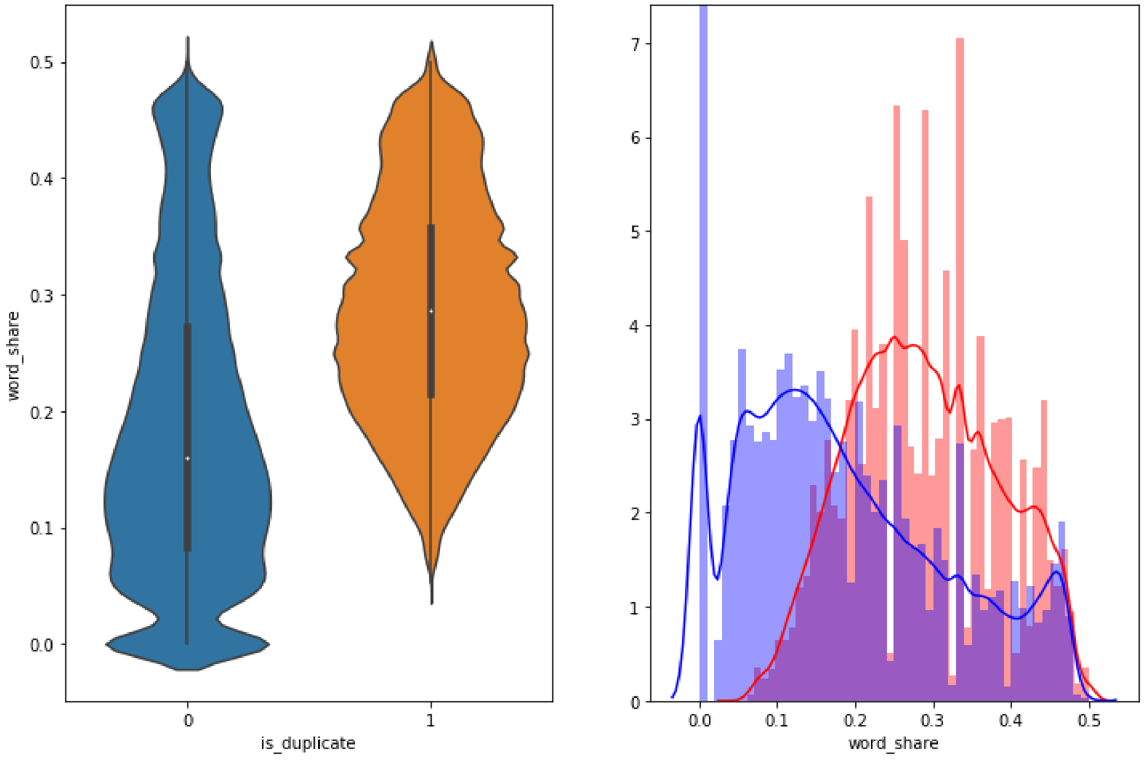
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

print ("Minimum length of the questions in question2 : " , min(df['q2_n_words']))

print ("Number of Questions with minimum length [question1] :", df[df['q1_n_words']== 1].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']== 1].shape[0])
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

In [16]:

```
import warnings
warnings.filterwarnings("ignore")

plt.figure(figsize=(12, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_share', data = df[0:])

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_share'][0:] , label = "1", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , color = 'blue' )
plt.show()
```

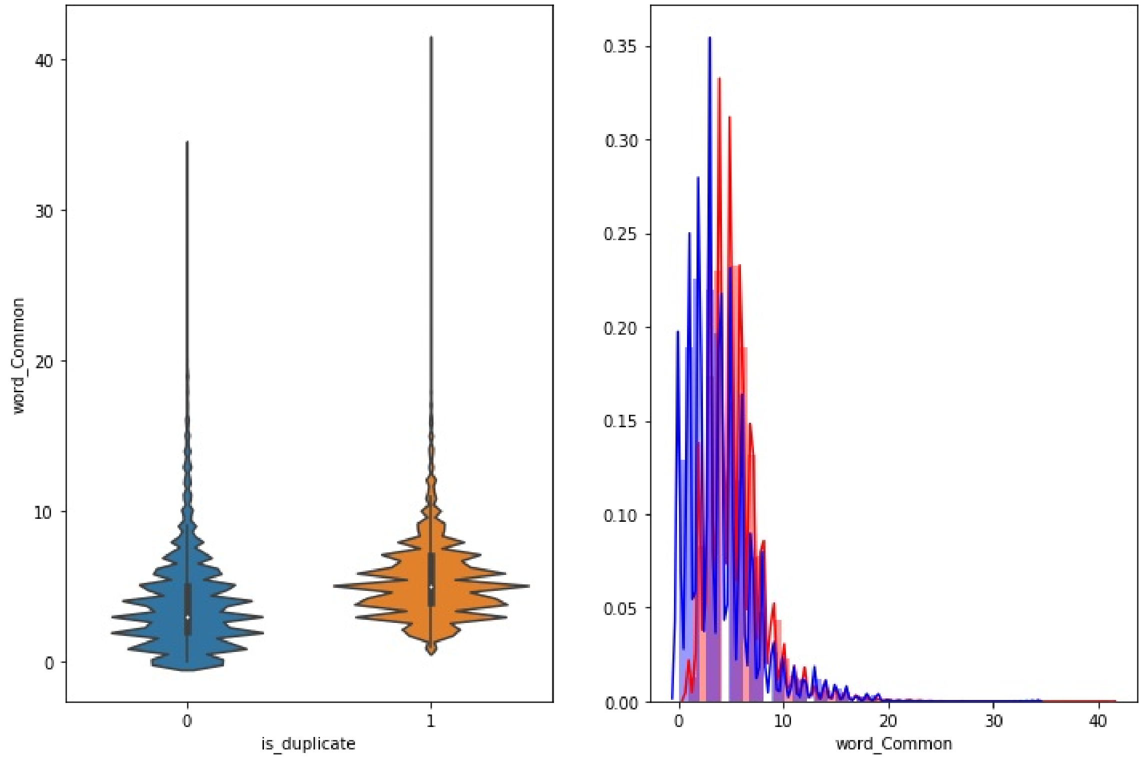


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [17]:

```
import warnings
warnings.filterwarnings("ignore")

plt.figure(figsize=(12, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_Common', data = df[0:])

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_Common'][0:] , label = "1", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label = "0" , color = 'blue' )
plt.show()
```



The distributions of the word_Common feature in similar and non-similar questions are highly overlapping

In [18]:

```
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

In [19]:

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byte-0x9c
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous notebook")
```

In [20]:

```
df.head(2)
```

Out[20]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_C
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14	12	10.0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	8	13	4.0

In [21]:

```
# To get the results in 4 decemal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("′", "'").replace("’", "'")\
                            .replace("won't", "will not").replace("cannot", "can not").replace("can'
", "can not")\
                            .replace("n't", " not").replace("what's", "what is").replace("it's", "it
is")\
                            .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
                            .replace("he's", "he is").replace("she's", "she is").replace("'s", " own
")\
                            .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar
")\
                            .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


    porter = PorterStemmer()
    pattern = re.compile('\W')

    if type(x) == type(''):
        x = re.sub(pattern, ' ', x)


    if type(x) == type(''):
        x = porter.stem(x)
        example1 = BeautifulSoup(x)
        x = example1.get_text()


    return x
```

In [22]:

```
def get_token_features(q1, q2):
    token_features = [0.0]*10

    # Converting the Sentence into Tokens:
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])

    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])

    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))

    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))

    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)

    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

    # First word of both question is same or not
    token_features[7] = int(q1_tokens[0] == q2_tokens[0])

    token_features[8] = abs(len(q1_tokens) - len(q2_tokens))

    #Average Token Length of both Questions
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

def extract_features(df):
    # preprocessing each question
    df["question1"] = df["question1"].fillna("").apply(preprocess)
    df["question2"] = df["question2"].fillna("").apply(preprocess)

    print("token features...")

    # Merging Features with dataset

    token_features = df.apply(lambda x: get_token_features(x["question1"], x["question2"]), axis=1)

    df["cwc_min"]       = list(map(lambda x: x[0], token_features))
    df["cwc_max"]       = list(map(lambda x: x[1], token_features))
    df["csc_min"]       = list(map(lambda x: x[2], token_features))
    df["csc_max"]       = list(map(lambda x: x[3], token_features))
    df["ctc_min"]       = list(map(lambda x: x[4], token_features))
    df["ctc_max"]       = list(map(lambda x: x[5], token_features))
    df["last_word_eq"]  = list(map(lambda x: x[6], token_features))
    df["first_word_eq"] = list(map(lambda x: x[7], token_features))
    df["abs_len_diff"]  = list(map(lambda x: x[8], token_features))
    df["mean_len"]      = list(map(lambda x: x[9], token_features))
    #Computing Fuzzy Features and Merging with Dataset

    # do read this blog: http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
    # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-function-to-compare-2-st
rings
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"], 
x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the tokens alpha
betically, and 
    # then joining them back into a string We then compare the transformed strings with a simple r
atio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"], 
x["question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), a
is=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"], 
x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["question1"], x["qu
estion2"]), axis=1)
    return df
```

In [23]:

```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[23]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	fi
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983	...	0.785709	0.0	1
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988	...	0.466664	0.0	1

2 rows × 21 columns

In [24]:

```
# UnicodeEncodeError: 'charmap' codec can't encode characters : 
https://stackoverflow.com/questions/27092833/unicodeencodeerror-charmap-codec-cant-encode-characte
rs

df_duplicate = df[df['is_duplicate'] == 1]
dfp_nonduplicate = df[df['is_duplicate'] == 0]

# Converting 2d array of q1 and q2 and flatten the array: like {{1,2},{3,4}} to {1,2,3,4}
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).flatten()

print ("Number of data points in class 1 (duplicate pairs) :",len(p))
print ("Number of data points in class 0 (non duplicate pairs) :",len(n))
```

```
#Saving the np array into a text file
np.savetxt('train_p.txt', p, delimiter=' ', fmt='%s')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s',encoding='utf-8')
```

```
Number of data points in class 1 (duplicate pairs) : 298526
Number of data points in class 0 (non duplicate pairs) : 510054
```

In [25]:

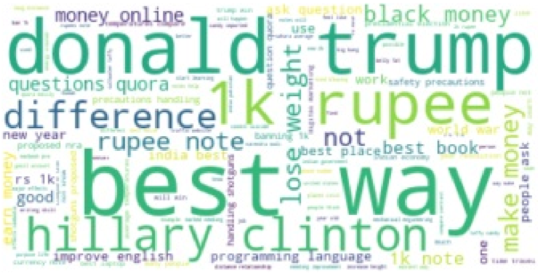
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

textp_w = open(path.join(d, 'train_p.txt')).read()
textn_w = open(path.join(d, 'train_n.txt')).read()
stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

stopwords.remove("no")
#stopwords.remove("good")
#stopwords.remove("love")
stopwords.remove("like")
#stopwords.remove("best")
#stopwords.remove("!")
print ("Total number of words in duplicate pair questions :",len(textp_w))
print ("Total number of words in non duplicate pair questions :",len(textn_w))
```

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33194892
```

In [26]:

```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
Word Cloud for Duplicate Question pairs
```



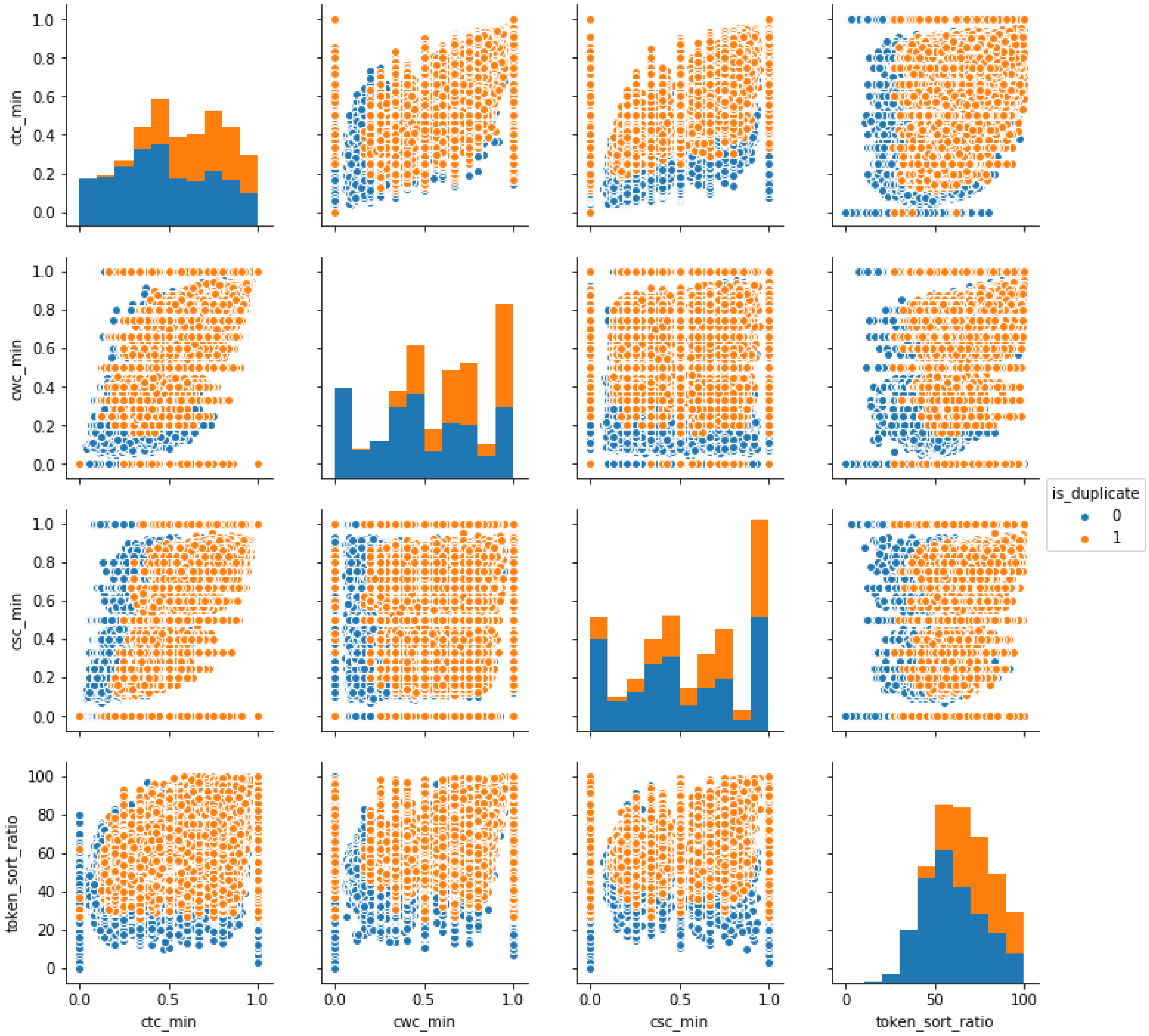
In [27]:

```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
Word Cloud for non-Duplicate Question pairs:
```

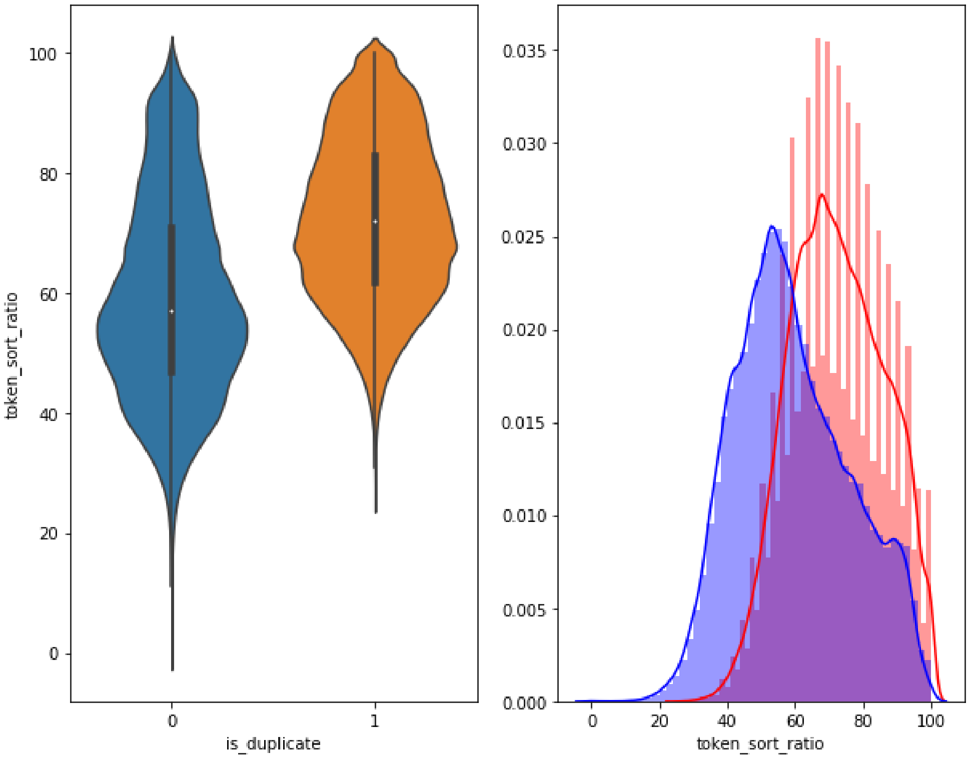
In [28]:

```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```



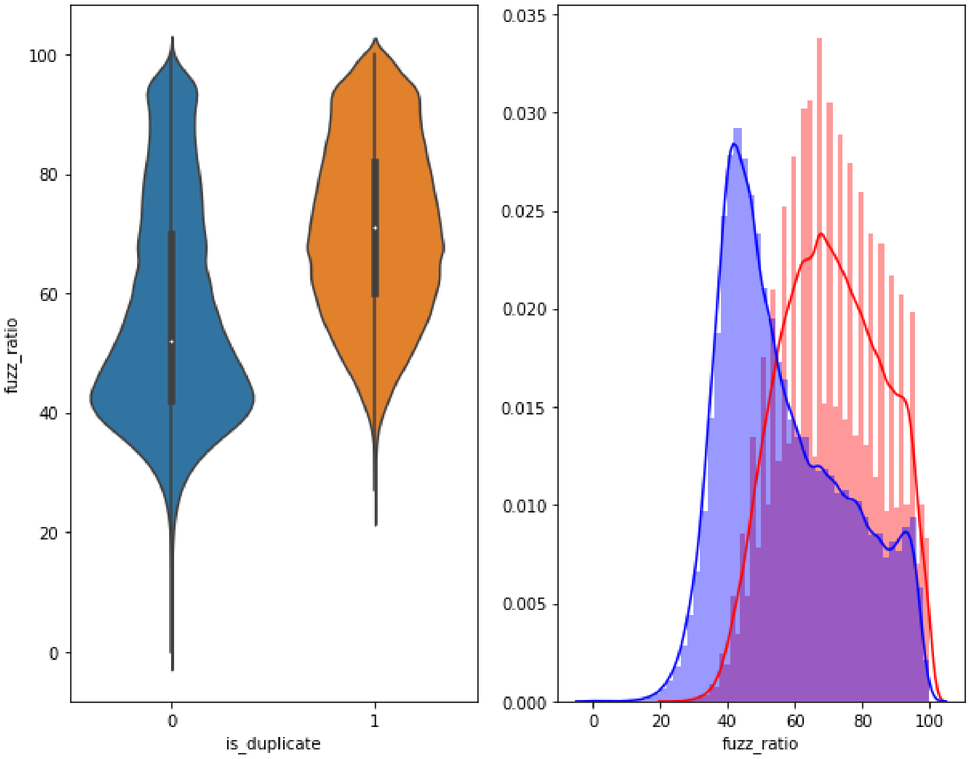
In [29]:

```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'token_sort_ratio', data = df[0:] , )

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['token_sort_ratio'][0:] , label = "1", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" , color = 'blue' )
plt.show()
```



In [30]:

```
plt.figure(figsize=(10, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'fuzz_ratio', data = df[0:] , )

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['fuzz_ratio'][0:] , label = "1", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , color = 'blue' )
plt.show()
```



In [31]:

```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 3
dimention
```

```
from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['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']])
y = dfp_subsampled['is_duplicate'].values
```

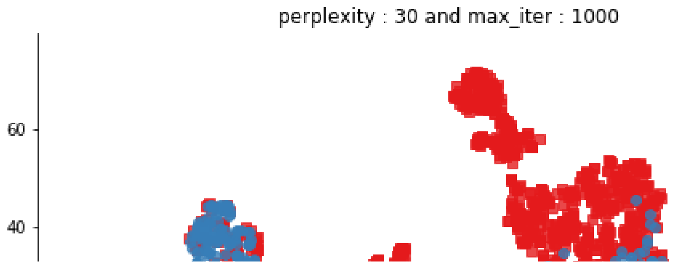
In [32]:

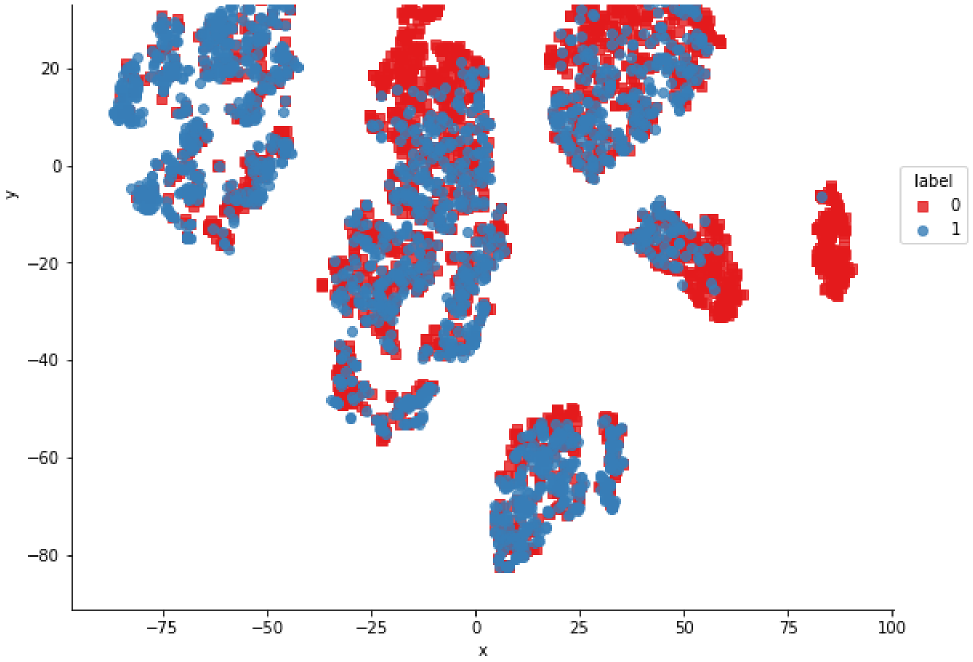
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.134s...
[t-SNE] Computed neighbors for 5000 samples in 0.427s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.235s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iterations in 2.679s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iterations in 2.018s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iterations in 1.946s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iterations in 1.977s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iterations in 1.990s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.273346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 iterations in 1.994s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iterations in 1.933s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iterations in 1.932s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iterations in 1.937s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iterations in 1.943s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iterations in 1.957s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iterations in 1.955s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iterations in 1.966s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iterations in 1.968s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iterations in 1.972s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iterations in 2.022s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iterations in 2.058s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iterations in 2.011s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iterations in 2.029s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iterations in 2.043s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [33]:

```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# draw the plot in appropriate place in the grid
sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",markers=['s','o
'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```





In [34]:

```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.010s...
[t-SNE] Computed neighbors for 5000 samples in 0.384s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.217s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 iterations in 10.307s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (50 iterations in 5.601s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (50 iterations in 4.891s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (50 iterations in 4.859s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (50 iterations in 4.808s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729782
[t-SNE] Iteration 300: error = 1.4978341, gradient norm = 0.0006938 (50 iterations in 6.005s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50 iterations in 7.778s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50 iterations in 7.937s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50 iterations in 7.928s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50 iterations in 7.785s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50 iterations in 7.581s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50 iterations in 7.628s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50 iterations in 7.662s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50 iterations in 7.699s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50 iterations in 7.722s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50 iterations in 7.684s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50 iterations in 7.701s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50 iterations in 7.661s)
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50 iterations in 7.664s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (50 iterations in 7.643s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [35]:

```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

data=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered features')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

In [36]:

```
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
from tqdm import tqdm
from tqdm import tqdm
```

In [37]:

```
# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy
```

In [38]:

```
# avoid decoding problems
df = pd.read_csv("train.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 ---------------------
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [39]:

```
df.head()
```

Out[39]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [40]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
import pickle

with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [43]:

```
X = df.drop('is_duplicate',axis=1)
y = df['is_duplicate']
```

In [45]:

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(df, y, stratify=y, test_size=0.3)
```

In [46]:

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

tfidf = TfidfVectorizer(lowercase=False )

tfidf.fit_transform(questions_train)
tfidf.transform(questions_test)


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

In [48]:

```
tfidf_w2v_q1_train = [];
for sentence in tqdm(list(X_train['question1'])): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_q1_train.append(vector)
```

```
100%|███████████████████████████████| 283003/283003 [00:13<00:00, 20386.33it/s]
```

In [49]:

```
tfidf_w2v_q1_test = [];
for sentence in tqdm(list(X_test['question1'])): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_q1_test.append(vector)
```

```
100%|███████████████████████████████| 121287/121287 [00:05<00:00, 20333.11it/s]
```

In [50]:

```
tfidf_w2v_q2_train = [];
for sentence in tqdm(list(X_train['question2'])): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_q2_train.append(vector)
    tfidf_w2v_q2_train.append(vector)
```

```
100%|████████████████████████████████| 283003/283003 [00:14<00:00, 19685.80it/s]
```

In [51]:

```
tfidf_w2v_q2_test = [];
for sentence in tqdm(list(X_test['question2'])): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_q2_test.append(vector)
```

```
100%|████████████████████████████████| 121287/121287 [00:06<00:00, 19575.05it/s]
```

In [53]:

```
df['q1_feats_m'] = list(tfidf_w2v_q1_train) + list(tfidf_w2v_q1_test)
df['q2_feats_m'] = list(tfidf_w2v_q2_train) + list(tfidf_w2v_q2_test)
```

In [54]:

```
df.shape
```

Out[54]:

```
(404290, 8)
```

In [55]:

```
df4 = df[:100000]
```

In [56]:

```
df4.shape
```

Out[56]:

```
(100000, 8)
```

In [57]:

```
# # en_vectors_web_lg, which includes over 1 million unique vectors.
# nlp = spacy.load('C://Users//Home//Anaconda//Lib//site-packages//en_core_web_sm//en_core_web_sm')

# vecs1 = []
# # https://github.com/noamraph/tqdm
# # tqdm is used to print the progress bar
# for qu1 in tqdm(list(df['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)
# df['q1_feats_m'] = list(vecs1)
```

In [58]:

```
# vecs2 = []
# for qu2 in tqdm(list(df['question2'])):
#     doc2 = nlp(qu2)
#     mean_vec2 = np.zeros([len(doc1), 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)
# df['q2_feats_m'] = list(vecs2)
```

In [59]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('nlp_features_train.csv'):
    dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous notebook")

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [60]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3_q1 = pd.DataFrame(df4.q1_feats_m.values.tolist(), index= df4.index)
df3_q2 = pd.DataFrame(df4.q2_feats_m.values.tolist(), index= df4.index)
```

In [61]:

```
# dataframe of nlp features
df1.head()
```

Out[61]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0

In [62]:

```
# data before preprocessing
```

```
df2.head()
```

Out[62]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	f
0	0	1	1	66	57	14	12	10.0	23.0	0.434783	2	0
1	1	4	1	51	88	8	13	4.0	20.0	0.200000	5	3
2	2	1	1	73	59	14	10	4.0	24.0	0.166667	2	0
3	3	1	1	50	65	11	9	0.0	19.0	0.000000	2	0
4	4	3	1	76	39	13	7	2.0	20.0	0.100000	4	2

In [63]:

```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[63]:

	0	1	2	3	4	5	6	7	8	9	...	290	29
0	-0.044626	0.083767	-0.106203	0.165566	-0.150050	0.243419	-3.330892	0.412070	-0.110667	-0.215905	...	-0.070985	-0.05980
1	0.292870	0.120320	0.021770	-0.231668	0.093190	0.096458	-3.170458	-0.223567	-0.367850	-0.372048	...	-0.006248	-0.00414
2	0.014788	-0.025796	-0.007299	-0.137170	-0.016655	0.021172	-3.535248	0.391330	0.078451	-0.315694	...	0.005157	-0.15545
3	0.150202	0.127466	0.049506	0.045332	0.151896	-0.086198	-3.327498	0.661608	0.093626	0.068638	...	0.188331	-0.00436
4	-0.146718	-0.046510	0.062985	-0.058413	-0.180394	-0.051723	-3.520295	-0.351013	-0.105945	-0.331911	...	-0.030364	-0.02948

5 rows × 300 columns

In [64]:

```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[64]:

	0	1	2	3	4	5	6	7	8	9	...	290	29
0	-0.151640	0.251848	-0.131989	-0.014701	-0.000463	0.259117	-3.995174	0.545616	-0.062063	-0.385017	...	-0.105307	-0.18743
1	0.180074	-0.085927	-0.005842	-0.091652	0.069622	0.020589	-3.296490	0.181048	-0.194256	-0.208682	...	0.037394	-0.06230
2	0.001079	-0.080343	-0.045628	-0.096480	0.021618	0.030643	-3.511330	0.342127	0.102384	-0.338135	...	0.027273	-0.14240
3	0.174858	0.146498	0.054115	0.050404	0.166474	-0.073931	-3.271321	0.663828	0.076910	0.103196	...	0.204116	0.02708
4	0.000096	0.087248	-0.042931	-0.140213	0.015480	0.056145	-3.982877	0.052960	-0.053332	-0.359104	...	-0.020834	-0.39811

5 rows × 300 columns

In [65]:

```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape[1])
print("Number of features in question2 w2v  dataframe :", df3_q2.shape[1])
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df3_q1.shape[1]+df3_q2.
shape[1])
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 300
Number of features in question2 w2v  dataframe : 300
Number of features in final dataframe  : 629
```

In [66]:

```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    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')
    result.to_csv('final_features.csv')
```

In [2]:

```
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
```

In [72]:

```
data = pd.read_csv('final_features.csv')
```

In [73]:

```
data = data[:100000]
```

In [74]:

```
data.head()
```

Out[74]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	86_y	
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	...	28.763412	- 10
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	...	117.588041	5.:
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	...	128.699924	17
3	3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	70.475263	-3
4	4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	...	18.852802	-7

5 rows × 221 columns

In [75]:

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

In [76]:

```
data.head()
```

Out[76]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	...	
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	...	11
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	...	12
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	...	70.
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	...	18.
5	0.666656	0.571420	0.888879	0.799992	0.705878	0.705878	1.0	0.0	0.0	17.0	...	13

5 rows × 218 columns

In [77]:

```
data.shape
```

Out[77]:

```
(99999, 218)
```

In [78]:

```
# 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)
```

In [79]:

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

In [80]:

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

In [81]:

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

```
Number of data points in train data : (69999, 218)
Number of data points in test data : (30000, 218)
```

In [82]:

```
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:  0.6274518207402963 Class 1:  0.3725481792597037
---------- Distribution of output variable in train data ----------
Class 0:  0.3725333333333333 Class 1:  0.3725333333333333
```

In [35]:

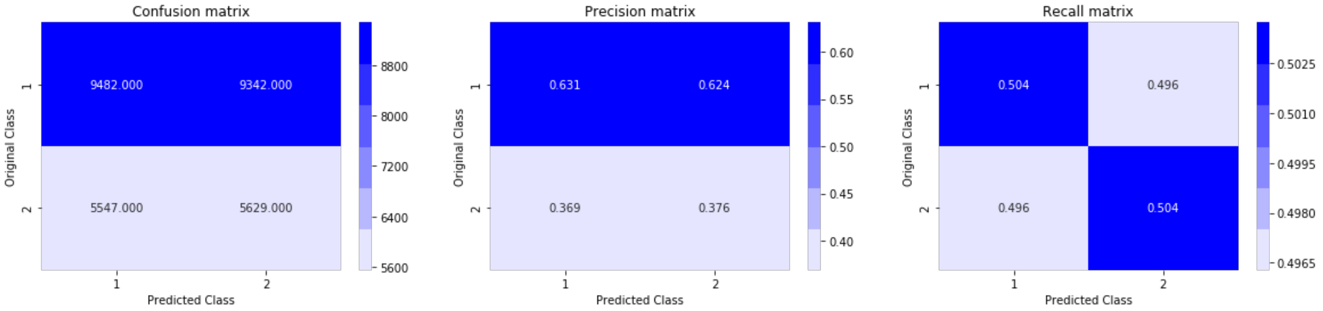
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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 [84]:

```
# 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 0.8838984971950752
```



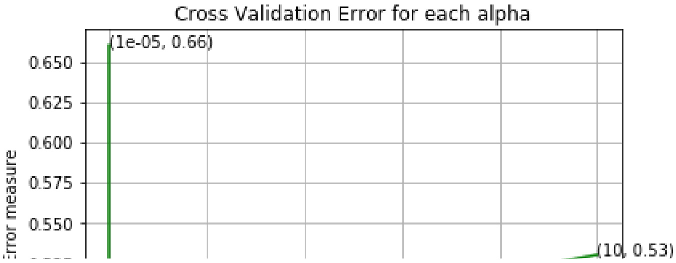
4.4 Logistic Regression with hyperparameter tuning

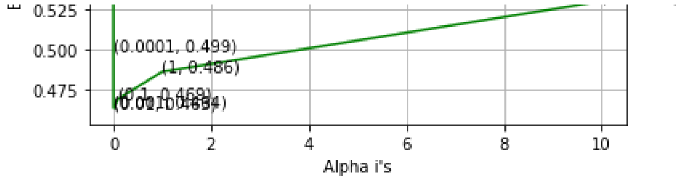
In [85]:

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

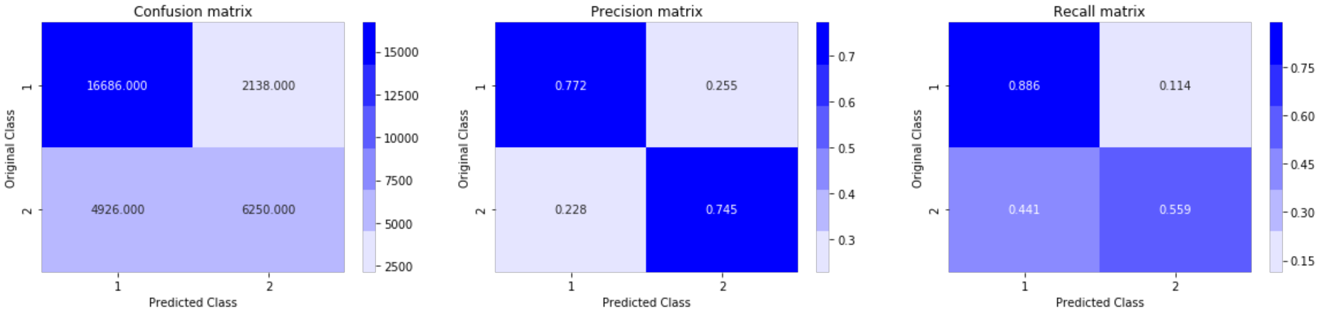
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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: 0.6602902114362172
For values of alpha =  0.0001 The log loss is: 0.4988492070150951
For values of alpha =  0.001 The log loss is: 0.46445085258437163
For values of alpha =  0.01 The log loss is: 0.4628610720359229
For values of alpha =  0.1 The log loss is: 0.4689527210298206
For values of alpha =  1 The log loss is: 0.48575368197407376
For values of alpha =  10 The log loss is: 0.529563953916008
```





```
For values of best alpha =  0.01 The train log loss is: 0.46235748817614986
For values of best alpha =  0.01 The test log loss is: 0.4628610720359229
Total number of data points : 30000
```



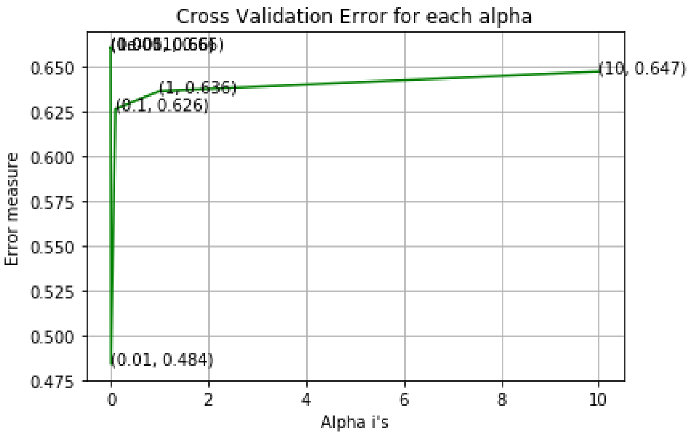
4.5 Linear SVM with hyperparameter tuning

In [86]:

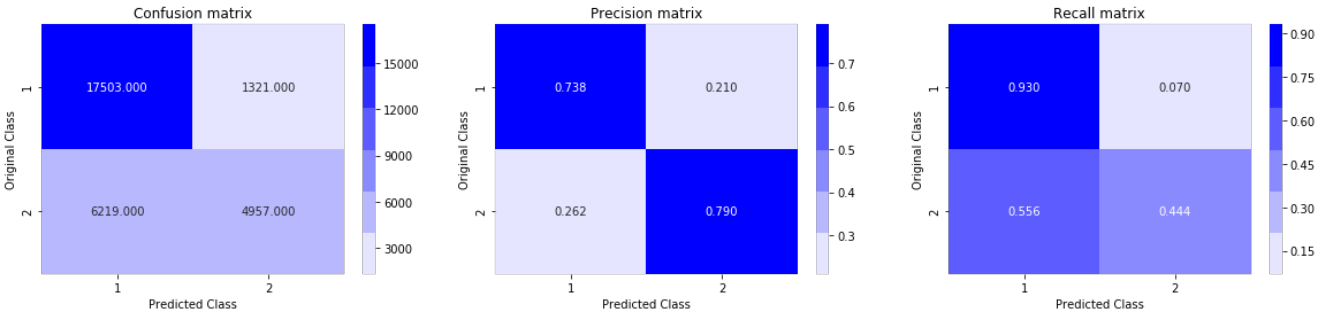
```python
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


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

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
```

```
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 0.6602902114362172
For values of alpha =  0.0001 The log loss is: 0.6602902114362172
For values of alpha =  0.001 The log loss is: 0.6602902114362172
For values of alpha =  0.01 The log loss is: 0.48373269023388743
For values of alpha =  0.1 The log loss is: 0.626028621115589
For values of alpha =  1 The log loss is: 0.636025348204608
For values of alpha =  10 The log loss is: 0.6469415950764378
```



```
For values of best alpha =  0.01 The train log loss is: 0.4844715021212608
For values of best alpha =  0.01 The test log loss is: 0.48373269023388743
Total number of data points : 30000
```



4.6 XGBoost

In [87]:

```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn import metrics   #Additional scklearn functions
from sklearn.model_selection import RandomizedSearchCV   #Perforing randomized search
```

In [88]:

```
# XGBoost Hyper-Parameter Tuning - https://www.analyticsvidhya.com/blog/2016/03/complete-guide-par
ameter-tuning-xgboost-with-codes-python/

params = {
  'n_estimators':[700,800,900,1000,1100],
  'learning_rate':[0.0001,0.001,0.01,0.1],
 'colsample_bytree':[0.1,0.2,0.4,0.5,0.6],
   'subsample': [0.1,0.2,0.4,0.5,0.6]
}
```

In [89]:

```
clf = RandomizedSearchCV(estimator = XGBClassifier(),param_distributions= params, scoring='neg_log_
loss',iid=False, cv=5,verbose= 5)


clf.fit(X_train, y_train)
# train_auc= clf.cv_results_['mean_train_score']
# train_auc_std= clf.cv_results_['std_train_score']
# cv_auc = clf.cv_results_['mean_test_score']
# cv_auc_std= clf.cv_results_['std_test_score']
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2, score=-0.671, t
otal= 3.3min
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.3min remaining:    0.0s
```

```
[CV] subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2, score=-0.671, t
otal= 3.3min
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  6.7min remaining:    0.0s
```

```
[CV] subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2, score=-0.671, t
otal= 3.3min
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 10.0min remaining:    0.0s
```

```
[CV] subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2, score=-0.671, t
otal= 3.3min
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 13.4min remaining:    0.0s
```

```
[CV] subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=800, learning_rate=0.0001, colsample_bytree=0.2, score=-0.671, t
otal= 3.3min
[CV] subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4, score=-0.365,
total= 3.3min
[CV] subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4, score=-0.373,
total= 3.3min
[CV] subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4, score=-0.359,
total= 3.3min
[CV] subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4, score=-0.373,
total= 3.3min
[CV] subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.1, n_estimators=800, learning_rate=0.1, colsample_bytree=0.4, score=-0.370,
total= 3.5min
[CV] subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4
[CV]  subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4, score=-0.374, tot
al= 3.7min
[CV] subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4
[CV]  subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4, score=-0.378, tot
al= 3.7min
```

```
[CV] subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4
[CV]  subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4, score=-0.372, tot
al= 3.7min
[CV] subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4
[CV]  subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4, score=-0.378, tot
al= 3.7min
[CV] subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4
[CV]  subsample=0.2, n_estimators=700, learning_rate=0.01, colsample_bytree=0.4, score=-0.378, tot
al= 3.5min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5, score=-0.501, t
otal= 4.9min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5, score=-0.500, t
otal= 4.5min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5, score=-0.499, t
otal= 4.5min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5, score=-0.502, t
otal= 4.5min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.001, colsample_bytree=0.5, score=-0.502, t
otal= 4.5min
[CV] subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6
[CV]  subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6, score=-0.669, t
otal= 6.8min
[CV] subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6
[CV]  subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6, score=-0.669, t
otal= 6.7min
[CV] subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6
[CV]  subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6, score=-0.669, t
otal= 6.6min
[CV] subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6
[CV]  subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6, score=-0.669, t
otal= 6.2min
[CV] subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6
[CV]  subsample=0.6, n_estimators=700, learning_rate=0.0001, colsample_bytree=0.6, score=-0.669, t
otal= 6.1min
[CV] subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5, score=-0.368, tot
al= 6.4min
[CV] subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5, score=-0.374, tot
al= 6.4min
[CV] subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5, score=-0.367, tot
al= 6.4min
[CV] subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5, score=-0.374, tot
al= 6.3min
[CV] subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.6, n_estimators=800, learning_rate=0.01, colsample_bytree=0.5, score=-0.373, tot
al= 6.3min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6, score=-0.534, to
tal= 6.0min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6, score=-0.533, to
tal= 5.9min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6, score=-0.532, to
tal= 6.0min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6, score=-0.534, to
tal= 6.0min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.6, score=-0.534, to
tal= 5.9min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4, score=-0.338,
total= 7.2min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4, score=-0.347,
total= 7.2min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4, score=-0.334,
total= 7.2min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4, score=-0.346,
total= 7.2min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.4, score=-0.342,
total= 7.2min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1, score=-0.671,
total= 2.9min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1, score=-0.671,
total= 2.9min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1, score=-0.671,
total= 2.9min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1, score=-0.671,
total= 2.9min
[CV] subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=1000, learning_rate=0.0001, colsample_bytree=0.1, score=-0.671,
total= 2.9min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5, score=-0.537, to
tal= 5.3min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5, score=-0.536, to
tal= 5.3min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5, score=-0.535, to
tal= 5.3min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5, score=-0.537, to
tal= 5.3min
[CV] subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=700, learning_rate=0.001, colsample_bytree=0.5, score=-0.537, to
tal= 5.3min
```

```
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed: 245.1min finished
```

Out[89]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0...
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid=False, n_iter=10, n_jobs=None,
                   param_distributions={'colsample_bytree': [0.1, 0.2, 0.4, 0.5,
                                                             0.6],
                                        'learning_rate': [0.0001, 0.001, 0.01,
                                                          0.1],
                                        'n_estimators': [700, 800, 900, 1000,
                                                         1100],
                                        'subsample': [0.1, 0.2, 0.4, 0.5, 0.6]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='neg_log_loss', verbose=5)
```

In [90]:

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
# params['max_depth'] = 4
params['subsample'] = 0.5
params['n_estimators'] = 700
```

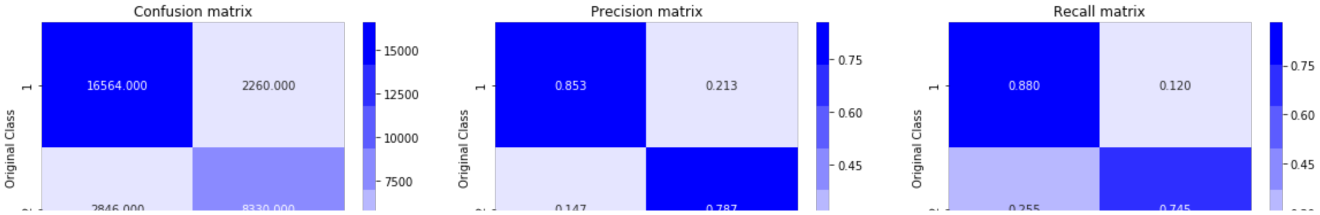
```
params['n_estimators'] = 700
params['learning_rate'] = 0.1
params['colsample_bytree'] = 0.1


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))
```

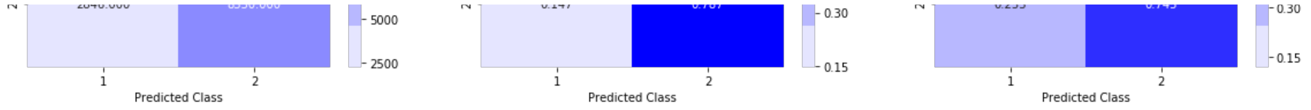
```
[0] train-logloss:0.677919 valid-logloss:0.678308
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10] train-logloss:0.529508 valid-logloss:0.532802
[20] train-logloss:0.457862 valid-logloss:0.464826
[30] train-logloss:0.420483 valid-logloss:0.431069
[40] train-logloss:0.403828 valid-logloss:0.41821
[50] train-logloss:0.387769 valid-logloss:0.405417
[60] train-logloss:0.37001 valid-logloss:0.391155
[70] train-logloss:0.356563 valid-logloss:0.380367
[80] train-logloss:0.344203 valid-logloss:0.371035
[90] train-logloss:0.336926 valid-logloss:0.366337
[100] train-logloss:0.32917 valid-logloss:0.362198
[110] train-logloss:0.321084 valid-logloss:0.357555
[120] train-logloss:0.314838 valid-logloss:0.354957
[130] train-logloss:0.310292 valid-logloss:0.353858
[140] train-logloss:0.304596 valid-logloss:0.351385
[150] train-logloss:0.299821 valid-logloss:0.349887
[160] train-logloss:0.295099 valid-logloss:0.348344
[170] train-logloss:0.290619 valid-logloss:0.347308
[180] train-logloss:0.286243 valid-logloss:0.345924
[190] train-logloss:0.283021 valid-logloss:0.345225
[200] train-logloss:0.278835 valid-logloss:0.344882
[210] train-logloss:0.274986 valid-logloss:0.344337
[220] train-logloss:0.270835 valid-logloss:0.343501
[230] train-logloss:0.267353 valid-logloss:0.343122
[240] train-logloss:0.263541 valid-logloss:0.342561
[250] train-logloss:0.259767 valid-logloss:0.34172
[260] train-logloss:0.256538 valid-logloss:0.341114
[270] train-logloss:0.253254 valid-logloss:0.340899
[280] train-logloss:0.250466 valid-logloss:0.340585
[290] train-logloss:0.247235 valid-logloss:0.34036
[300] train-logloss:0.244381 valid-logloss:0.340026
[310] train-logloss:0.241874 valid-logloss:0.339852
[320] train-logloss:0.239147 valid-logloss:0.339599
[330] train-logloss:0.235939 valid-logloss:0.339522
[340] train-logloss:0.233358 valid-logloss:0.339709
Stopping. Best iteration:
[327] train-logloss:0.236704 valid-logloss:0.339483

The test log loss is: 0.3395472340912241
```

In [91]:

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

```
Total number of data points : 30000
```





Assignments

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

In [3]:

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

In [4]:

```
data.head()
```

Out[4]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14	12	10.0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	8	13	4.0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	14	10	4.0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65	11	9	0.0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	13	7	2.0

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 17 columns):
```

```
Data columns (total 17 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
freq_qid1       404290 non-null int64
freq_qid2       404290 non-null int64
q1len           404290 non-null int64
q2len           404290 non-null int64
q1_n_words      404290 non-null int64
q2_n_words      404290 non-null int64
word_Common     404290 non-null float64
word_Total      404290 non-null float64
word_share      404290 non-null float64
freq_q1+q2      404290 non-null int64
freq_q1-q2      404290 non-null int64
dtypes: float64(3), int64(12), object(2)
memory usage: 52.4+ MB
```

In [6]:

```
data.columns
```

Out[6]:

```
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 [7]:

```
data2 = pd.read_csv('nlp_features_train.csv',encoding='latin-1')
```

In [8]:

```
data2.columns
```

Out[8]:

```
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')
```

In [9]:

```
#Dropping common columns from data2

data2 = data2.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [10]:

```
data2.head()
```

Out[10]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	tok
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	10
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	86
2	2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	66
3	3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	36
4	4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	67

In [11]:

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 16 columns):
id                        404290 non-null int64
cwc_min                   404290 non-null float64
cwc_max                   404290 non-null float64
csc_min                   404290 non-null float64
csc_max                   404290 non-null float64
ctc_min                   404290 non-null float64
ctc_max                   404290 non-null float64
last_word_eq              404290 non-null float64
first_word_eq             404290 non-null float64
abs_len_diff              404290 non-null float64
mean_len                  404290 non-null float64
token_set_ratio           404290 non-null int64
token_sort_ratio          404290 non-null int64
fuzz_ratio                404290 non-null int64
fuzz_partial_ratio        404290 non-null int64
longest_substr_ratio      404290 non-null float64
dtypes: float64(11), int64(5)
memory usage: 49.4 MB
```

In [12]:

```
dataset = pd.merge(data,data2,on='id')
```

In [13]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404290 entries, 0 to 404289
Data columns (total 32 columns):
id                        404290 non-null int64
qid1                      404290 non-null int64
qid2                      404290 non-null int64
question1                 404289 non-null object
question2                 404288 non-null object
is_duplicate              404290 non-null int64
freq_qid1                 404290 non-null int64
freq_qid2                 404290 non-null int64
q1len                     404290 non-null int64
q2len                     404290 non-null int64
q1_n_words                404290 non-null int64
q2_n_words                404290 non-null int64
word_Common               404290 non-null float64
word_Total                404290 non-null float64
word_share                404290 non-null float64
freq_q1+q2                404290 non-null int64
freq_q1-q2                404290 non-null int64
cwc_min                   404290 non-null float64
cwc_max                   404290 non-null float64
csc_min                   404290 non-null float64
csc_max                   404290 non-null float64
ctc_min                   404290 non-null float64
ctc_max                   404290 non-null float64
last_word_eq              404290 non-null float64
first_word_eq             404290 non-null float64
abs_len_diff              404290 non-null float64
mean_len                  404290 non-null float64
token_set_ratio           404290 non-null int64
token_sort_ratio          404290 non-null int64
fuzz_ratio                404290 non-null int64
fuzz_partial_ratio        404290 non-null int64
longest_substr_ratio      404290 non-null float64
dtypes: float64(14), int64(16), object(2)
memory usage: 101.8+ MB
```

In [14]:

```
# question1,question2 contain null values, therefore dropping null values

dataset = dataset.dropna()
```

In [15]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404287 entries, 0 to 404289
Data columns (total 32 columns):
id                        404287 non-null int64
qid1                      404287 non-null int64
qid2                      404287 non-null int64
question1                 404287 non-null object
question2                 404287 non-null object
is_duplicate              404287 non-null int64
freq_qid1                 404287 non-null int64
freq_qid2                 404287 non-null int64
q1len                     404287 non-null int64
q2len                     404287 non-null int64
q1_n_words                404287 non-null int64
q2_n_words                404287 non-null int64
word_Common               404287 non-null float64
word_Total                404287 non-null float64
word_share                404287 non-null float64
freq_q1+q2                404287 non-null int64
freq_q1-q2                404287 non-null int64
cwc_min                   404287 non-null float64
cwc_max                   404287 non-null float64
csc_min                   404287 non-null float64
csc_max                   404287 non-null float64
ctc_min                   404287 non-null float64
ctc_max                   404287 non-null float64
last_word_eq              404287 non-null float64
first_word_eq             404287 non-null float64
abs_len_diff              404287 non-null float64
mean_len                  404287 non-null float64
token_set_ratio           404287 non-null int64
token_sort_ratio          404287 non-null int64
fuzz_ratio                404287 non-null int64
fuzz_partial_ratio        404287 non-null int64
longest_substr_ratio      404287 non-null float64
dtypes: float64(14), int64(16), object(2)
memory usage: 101.8+ MB
```

In [16]:

```
dataset.head()
```

Out[16]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	...	ctc_max	last_word_eq	first_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	...	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	4	1	51	88	...	0.466664	0.0	1.0

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	...	ctc_max	last_word_eq	first_
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	...	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	1	1	50	65	...	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	3	1	76	39	...	0.307690	0.0	1.0

5 rows × 32 columns

In [17]:

```
dataset.columns
```

Out[17]:

```
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',
       '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 [18]:

```
len(dataset.columns)
```

Out[18]:

```
32
```

In [19]:

```
X = dataset.drop('is_duplicate',axis=1)
y = dataset['is_duplicate']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

In [20]:

```
dataset = dataset.drop('is_duplicate',axis=1)
```

In [21]:

```
dataset.columns
```

Out[21]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', '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'],
      dtype='object')
```

Applying TFIDF Vectorizer

In [22]:

```
dataset.columns
```

Out[22]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', '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'],
      dtype='object')
```

In [23]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['question1'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_q1_tfidf = vectorizer.transform(X_train['question1'].values)
X_test_q1_tfidf = vectorizer.transform(X_test['question1'].values)

v8 = vectorizer.get_feature_names()

print(X_train_q1_tfidf.shape)
print(X_test_q1_tfidf.shape)
```

```
(270872, 56790)
(133415, 56790)
```

In [24]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train ['question2'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_q2_tfidf = vectorizer.transform(X_train['question2'].values)
X_test_q2_tfidf = vectorizer.transform(X_test['question2'].values)

v9 = vectorizer.get_feature_names()

print(X_train_q2_tfidf.shape)
print(X_test_q2_tfidf.shape)
```

```
(270872, 52426)
(133415, 52426)
```

In [25]:

```
X_train = X_train.drop(['question1','question2'],axis=1)
X_test = X_test.drop(['question1','question2'],axis=1)
```

In [26]:

```
X_train = hstack((X_train,X_train_q1_tfidf,X_train_q2_tfidf))
X_test = hstack((X_test,X_test_q1_tfidf,X_test_q2_tfidf))
```

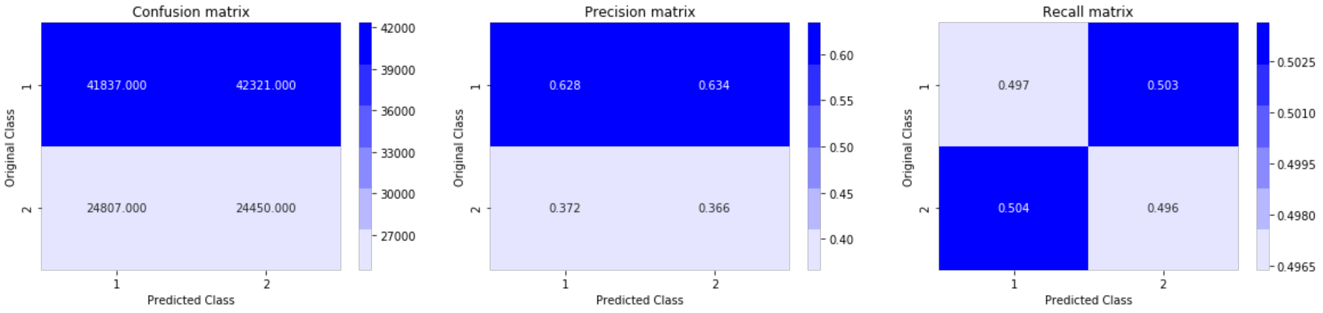
Random Model

Random Model

In [116]:

```
# 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((len(y_test),2))
for i in range(len(y_test)):
    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 0.8900794921900257
```

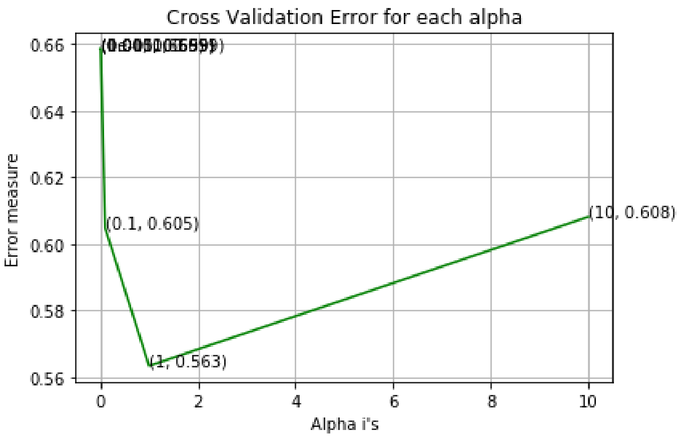


Logistic Regression

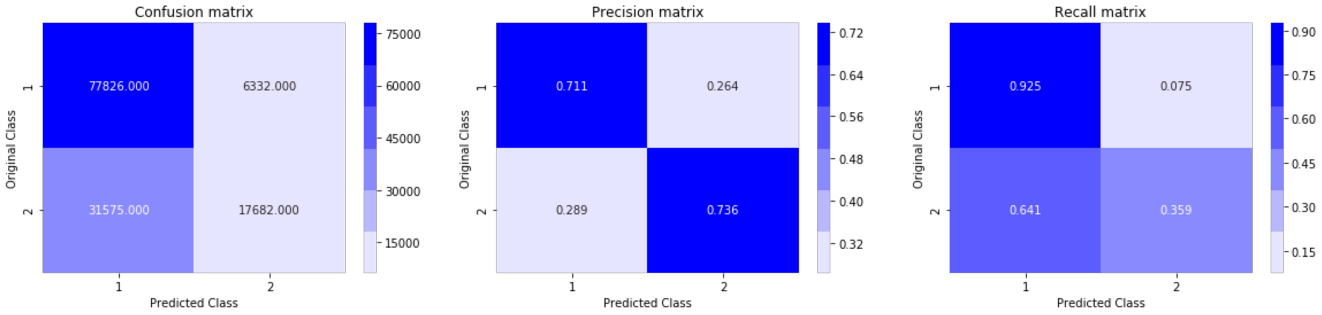
In [117]:

```
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, 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: 0.6585292647392701
For values of alpha =  0.0001 The log loss is: 0.6585292647392701
For values of alpha =  0.001 The log loss is: 0.6585292647392701
For values of alpha =  0.01 The log loss is: 0.6585292647392701
For values of alpha =  0.1 The log loss is: 0.6048179153136248
For values of alpha =  1 The log loss is: 0.5634430430946619
For values of alpha =  10 The log loss is: 0.6081049668986148
```



```
For values of best alpha =  1 The train log loss is: 0.5633375823816381
For values of best alpha =  1 The test log loss is: 0.5634430430946619
Total number of data points : 133415
```

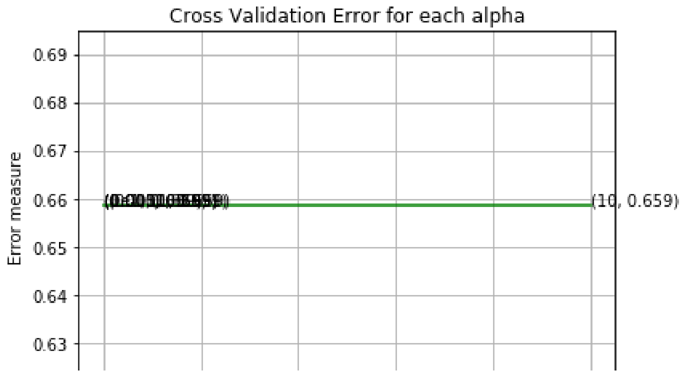


Linear SVM

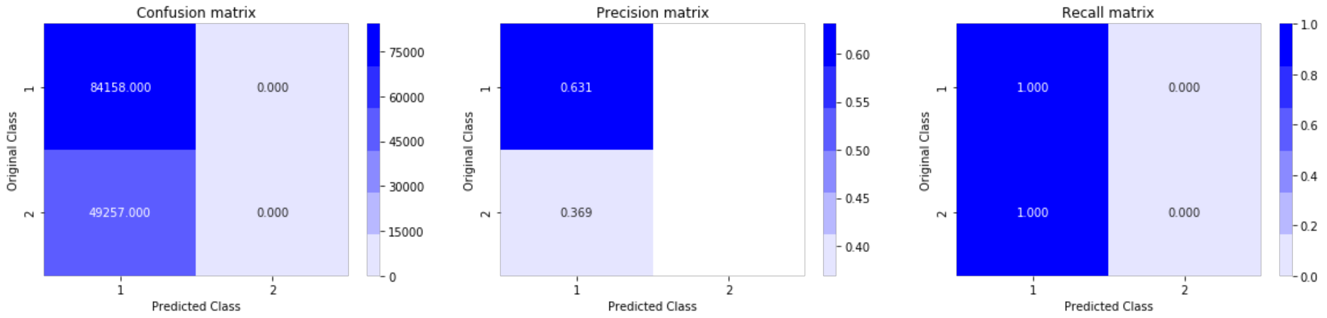
In [118]:

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

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

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

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


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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 0.6585292647392701
For values of alpha =  0.0001 The log loss is: 0.6585292647392701
For values of alpha =  0.001 The log loss is: 0.6585292647392701
For values of alpha =  0.01 The log loss is: 0.6585292647392701
For values of alpha =  0.1 The log loss is: 0.6585292647392701
For values of alpha =  1 The log loss is: 0.6585292647392701
For values of alpha =  10 The log loss is: 0.6585292647392701
```



```
For values of best alpha =  1e-05 The train log loss is: 0.6585286479436855
For values of best alpha =  1e-05 The test log loss is: 0.6585292647392701
Total number of data points : 133415
```



XG Boost with Hyper-parameter tuning

In [27]:

```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn import metrics   #Additional scklearn functions
from sklearn.model_selection import RandomizedSearchCV   #Perforing randomized search
```

In [30]:

```
# XGBoost Hyper-Parameter Tuning - https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

params = {
  'n_estimators':[1000,1100],
  'learning_rate':[0.001,0.01,0.1],
 'colsample_bytree':[0.1,0.2,0.3],
   'subsample': [0.4,0.5,0.6]
}
```

In [31]:

```
clf = RandomizedSearchCV(estimator = XGBClassifier(),param_distributions= params, scoring='neg_log_loss',iid=False, cv=5,verbose= 5)


clf.fit(X_train, y_train)
# train_auc= clf.cv_results_['mean_train_score']
# train_auc_std= clf.cv_results_['std_z train_score']
# cv_auc = clf.cv_results_['mean_test_score']
# cv_auc_std= clf.cv_results_['std_test_score']
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3, score=-0.312, total= 8.6min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  8.6min remaining:    0.0s
```

```
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3, score=-0.316, total= 8.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 16.7min remaining:    0.0s
```

```
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3, score=-0.316,
total= 8.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 24.8min remaining:    0.0s
```

```
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3, score=-0.317,
total= 8.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 32.9min remaining:    0.0s
```

```
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.3, score=-0.315,
total= 8.1min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.541, t
otal= 5.0min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.538, t
otal= 5.0min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.538, t
otal= 5.0min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.539, t
otal= 5.0min
[CV] subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.539, t
otal= 5.2min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3, score=-0.359, to
tal= 9.4min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3, score=-0.358, to
tal= 9.3min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3, score=-0.359, to
tal= 8.3min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3, score=-0.361, to
tal= 8.3min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.3, score=-0.360, to
tal= 8.3min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2, score=-0.314,
total= 6.1min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2, score=-0.317,
total= 6.1min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2, score=-0.317,
total= 6.1min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2, score=-0.318,
total= 6.1min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.2, score=-0.317,
total= 6.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1, score=-0.317,
total= 4.9min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1, score=-0.319,
total= 4.9min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1, score=-0.319,
total= 4.9min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1, score=-0.321,
total= 4.9min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1
```

```
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.1, colsample_bytree=0.1, score=-0.319,
total= 4.9min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2, score=-0.364, to
tal= 5.7min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2, score=-0.364, to
tal= 5.7min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2, score=-0.366, to
tal= 5.7min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2, score=-0.367, to
tal= 5.7min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.2, score=-0.366, to
tal= 5.7min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3, score=-0.362, to
tal= 7.0min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3, score=-0.360, to
tal= 7.0min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3, score=-0.362, to
tal= 7.0min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3, score=-0.363, to
tal= 7.0min
[CV] subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.4, n_estimators=1000, learning_rate=0.01, colsample_bytree=0.3, score=-0.363, to
tal= 7.0min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.541, t
otal= 5.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.538, t
otal= 5.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.538, t
otal= 5.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.539, t
otal= 5.1min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.539, t
otal= 5.1min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.541, t
otal= 4.9min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.538, t
otal= 4.9min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.538, t
otal= 4.9min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.539, t
otal= 4.9min
[CV] subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1
[CV]  subsample=0.4, n_estimators=1100, learning_rate=0.001, colsample_bytree=0.1, score=-0.539, t
otal= 4.9min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1, score=-0.377, to
tal= 5.0min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1, score=-0.374, to
tal= 5.0min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1, score=-0.376, to
tal= 5.0min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1, score=-0.378, to
tal= 5.0min
[CV] subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1
[CV]  subsample=0.6, n_estimators=1100, learning_rate=0.01, colsample_bytree=0.1, score=-0.377, to
tal= 5.0min
```

```
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed: 303.9min finished
```

Out[31]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           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),
                   iid=False, n_iter=10, n_jobs=None,
                   param_distributions={'colsample_bytree': [0.1, 0.2, 0.3],
                                        'learning_rate': [0.001, 0.01, 0.1],
                                        'n_estimators': [1000, 1100],
                                        'subsample': [0.4, 0.5, 0.6]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='neg_log_loss', verbose=5)
```

In [32]:

```
clf.best_params_
```

Out[32]:

```
{'subsample': 0.6,
 'n_estimators': 1100,
 'learning_rate': 0.1,
 'colsample_bytree': 0.3}
```

In [33]:

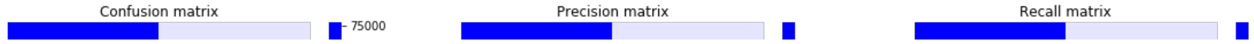
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['subsample'] = 0.6
params['n_estimators'] = 1100
params['learning_rate'] = 0.1
params['colsample_bytree'] = 0.3

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=False)

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))
```

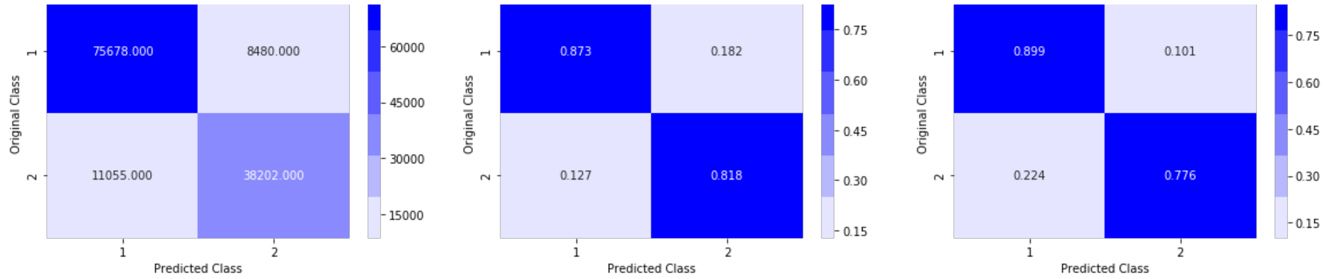
```
The test log loss is: 0.30876284908541185
```

In [36]:

```
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 : 133415
```





In [38]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model","Log-Loss"]

x.add_row(["TFIDF_W2V", "Random Model",0.89])
x.add_row(["TFIDF_W2V", "Logistic Regression",0.46])
x.add_row(["TFIDF_W2V", "Linear SVM", 0.48])
x.add_row(["TFIDF_W2V", "XG Boost",0.33])
print(x)


x = PrettyTable()
x.field_names = ["Vectorizer", "Model","Log-Loss"]

x.add_row(["TFIDF", "Random Model",0.88])
x.add_row(["TFIDF", "Logistic Regression",0.56])
x.add_row(["TFIDF", "Linear SVM", 0.65])
x.add_row(["TFIDF", "XG Boost",0.308])
print(x)
```

```
+------------+---------------------+----------+
| Vectorizer |        Model        | Log-Loss |
+------------+---------------------+----------+
| TFIDF_W2V  |     Random Model    |   0.89   |
| TFIDF_W2V  | Logistic Regression |   0.46   |
| TFIDF_W2V  |      Linear SVM     |   0.48   |
| TFIDF_W2V  |       XG Boost      |   0.33   |
+------------+---------------------+----------+
+------------+---------------------+----------+
| Vectorizer |        Model        | Log-Loss |
+------------+---------------------+----------+
|   TFIDF    |     Random Model    |   0.88   |
|   TFIDF    | Logistic Regression |   0.56   |
|   TFIDF    |      Linear SVM     |   0.65   |
|   TFIDF    |       XG Boost      |  0.308   |
+------------+---------------------+----------+
```

Inference :

Use TFIDF_W2v vectorizer for Logistic Regression and Linear SVM Models and TFIDF Vectorizer for XG Boost CLassifier models for minimum log-loss