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 st
ep 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 Indi
an 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 comme
nts?","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]:

```
!pip install distance
```

```
Requirement already satisfied: distance in c:\users\bolua\appdata\local\continuum\anaconda3\lib\site-pa
ckages (0.1.3)
```

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
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
import re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
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

from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import TfidfVectorizer

import sys
import os
from tqdm import tqdm

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

from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
import seaborn as sns

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.calibration import CalibratedClassifierCV
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarning:

Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning
```

3.1 Reading data and basic stats

In [3]:

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

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

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

In [4]:

```
df.head()
```

Out[4]:

	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 [5]:

```
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

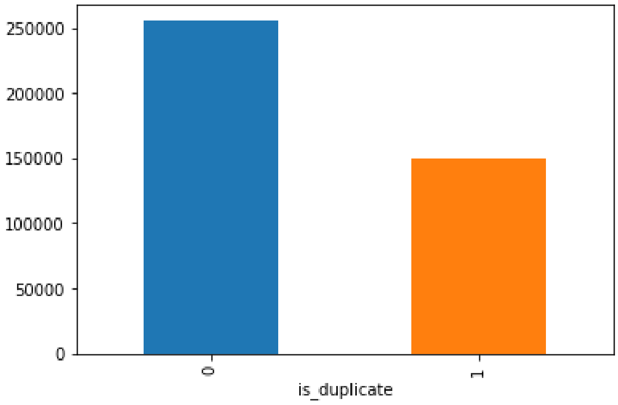
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [6]:

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

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b3bd7b4cc0>
```



In [7]:

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

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

In [8]:

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

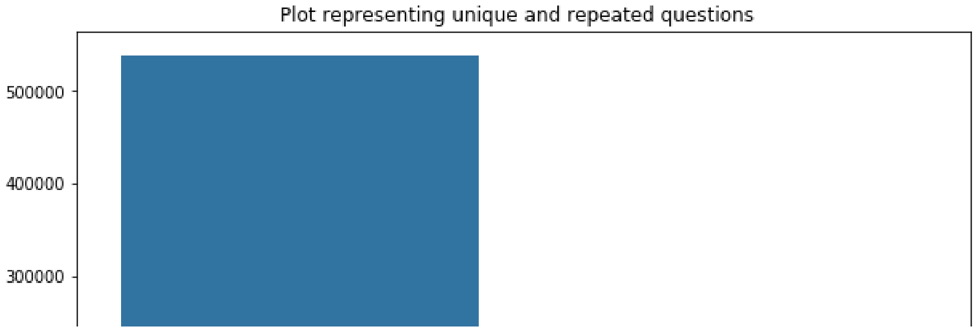
In [9]:

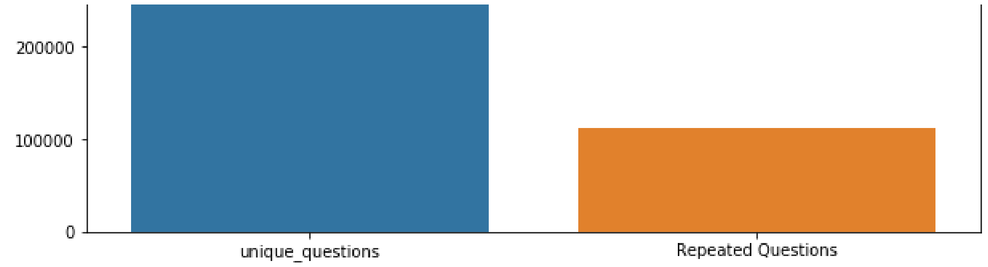
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(qs_morethan_oneti
me,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 [10]:

```
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 [11]:

```
#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])
```

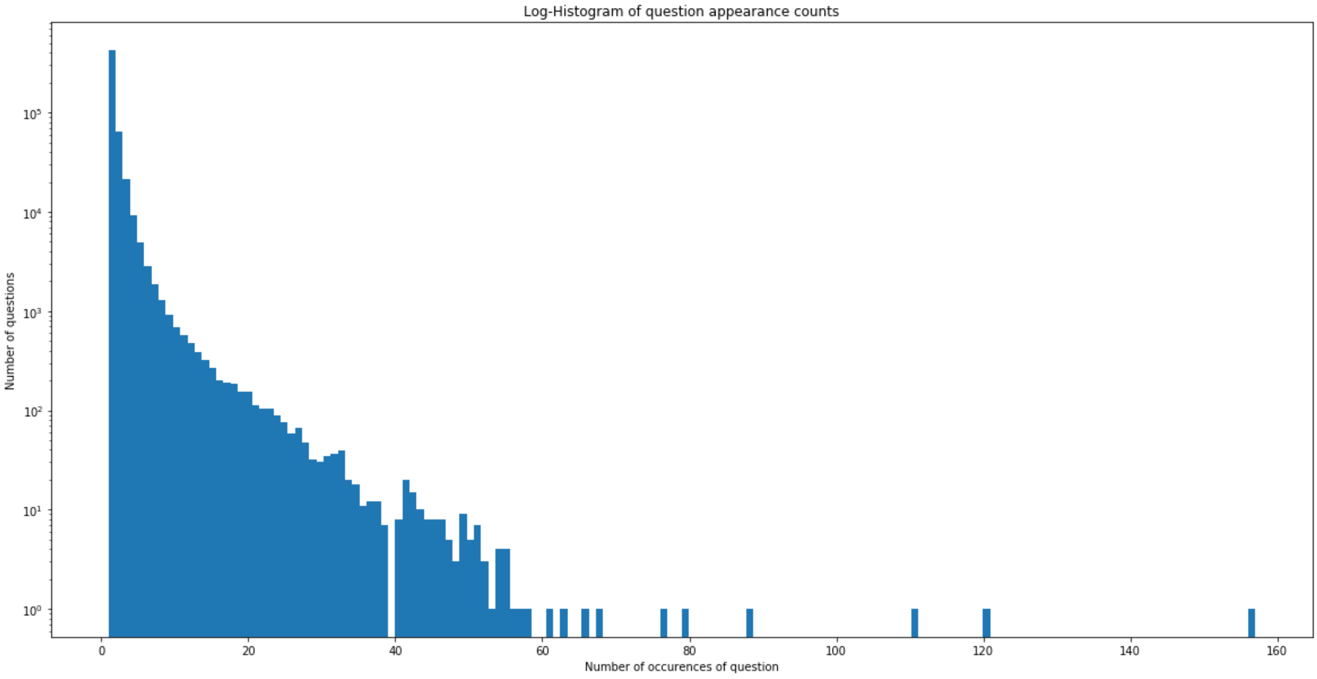
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [12]:

```
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 [13]:

```
#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 [14]:

```
# 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 [15]:

```
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[15]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common
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

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [16]:

```
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])
```

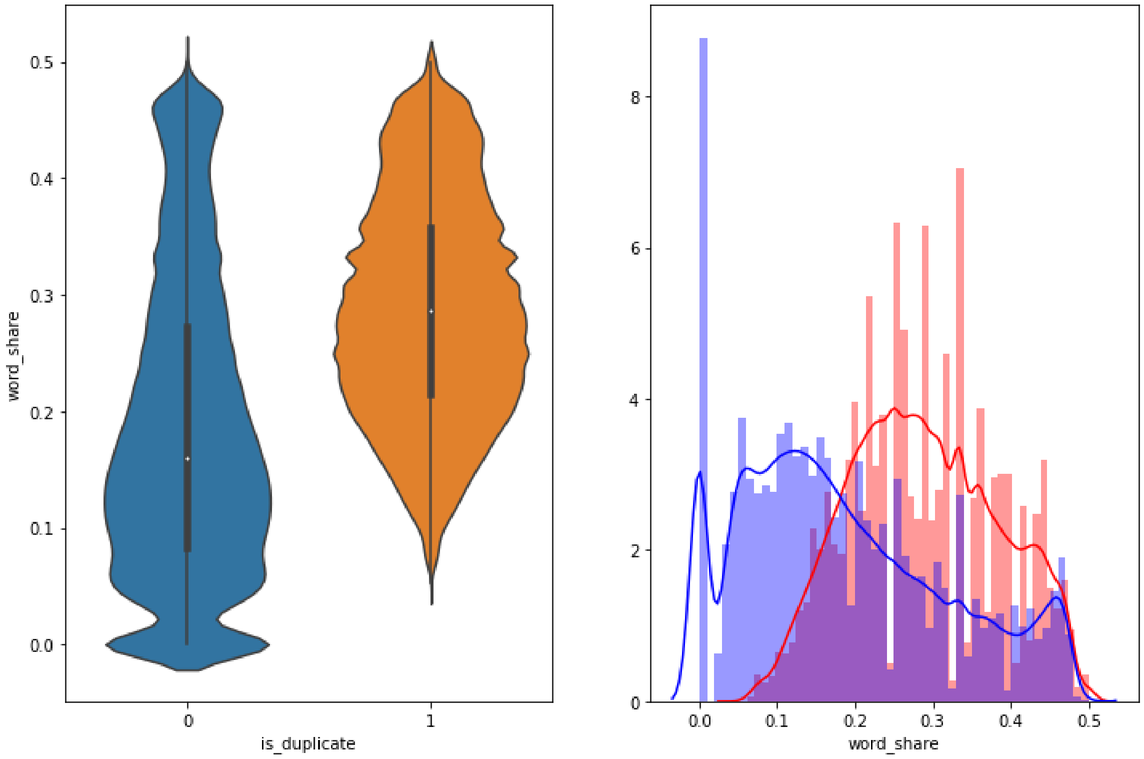
```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
```

```
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

In [17]:

```
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()
```

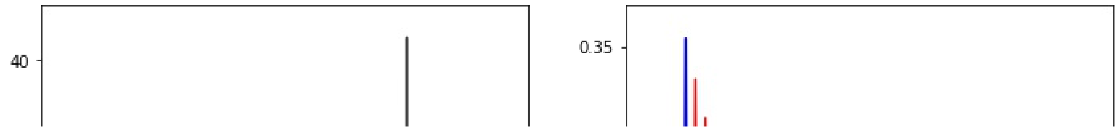


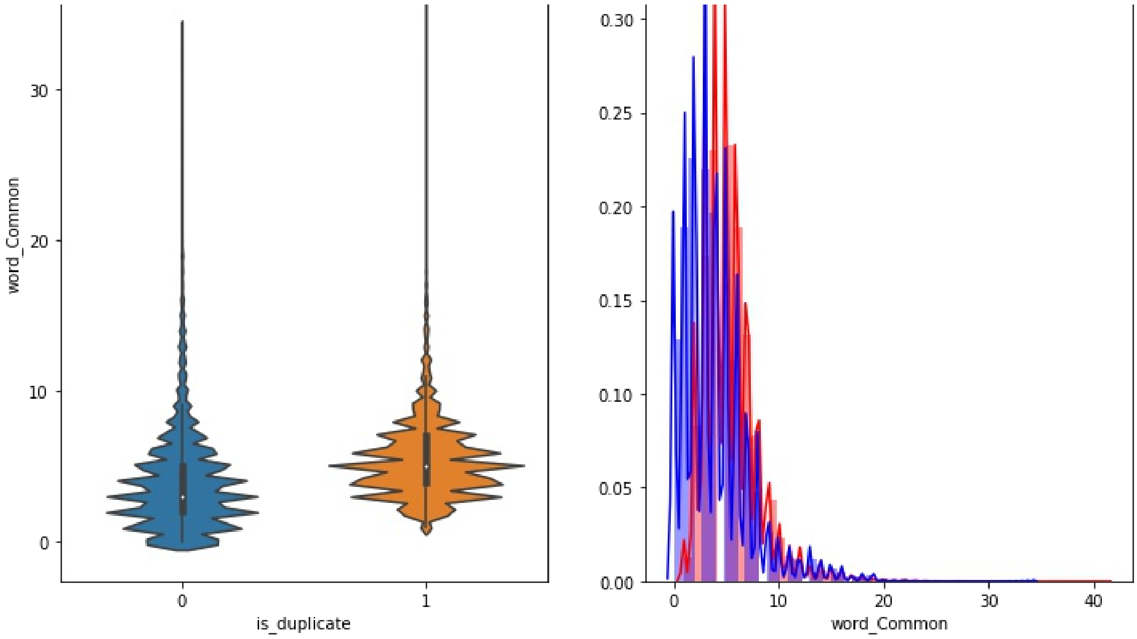
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [18]:

```
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 [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_Common	v
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	

3.4 Preprocessing of Text

- Preprocessing:
  - Removing html tags
  - Removing Punctuations
  - Performing stemming
  - Removing Stopwords
  - Expanding contractions etc.

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't",
"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
```

- Function to Compute and get the features : With 2 parameters of Question 1 and Question 2

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

- **Token**: You get a token by splitting sentence a space
- **Stop_Word** : stop words as per NLTK.
- **Word** : A token that is not a stop_word

Features:

- **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2
  cwc_min = common_word_count / (min(len(q1_words), len(q2_words))

- **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
  cwc_max = common_word_count / (max(len(q1_words), len(q2_words))

- **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2
  csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))

- **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))

- **ctc_min** : Ratio of common_token_count to min lenghth of token count of Q1 and Q2
  ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))

- **ctc_max** : Ratio of common_token_count to max lenghth of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

- **last_word_eq** : Check if First word of both questions is equal or not

- **last_word_eq** : Check if First word of both questions is equal or not
  last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])

- **first_word_eq** : Check if First word of both questions is equal or not
  first_word_eq = int(q1_tokens[0] == q2_tokens[0])

- **abs_len_diff** : Abs. length difference
  abs_len_diff = abs(len(q1_tokens) - len(q2_tokens))

- **mean_len** : Average Token Length of both Questions
  mean_len = (len(q1_tokens) + len(q2_tokens))/2

- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

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-strings
    # 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 alphabetic
ally, and
    # then joining them back into a string We then compare the transformed strings with a simple ratio(
).
    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"]), axis=
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["questi
on2"]), axis=1)
    return df
```

In [23]:

```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='utf-8')
    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	first_wor
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	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	

2 rows × 21 columns

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

- Creating Word Cloud of Duplicates and Non-Duplicates Question pairs
- We can observe the most frequent occuring words

In [24]:

```
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()

#items = [x.encode('utf-8') for x in p]
#array_unicode = np.array(items) # remove the brackets for line breaks

#index= [16616,16617,28598,28599,28596,28595,28594]
#p= np.delete(p,index)
#p=np.array(p,dtype=np.unicode)
#p.drop(index=16617)
#print(p[16617])

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', encoding = 'utf-8')
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'),encoding='utf-8').read()
textn_w = open(path.join(d, 'train_n.txt'),encoding='utf-8').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 : 33193130
```

Word Clouds generated from duplicate pair question's text

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
```

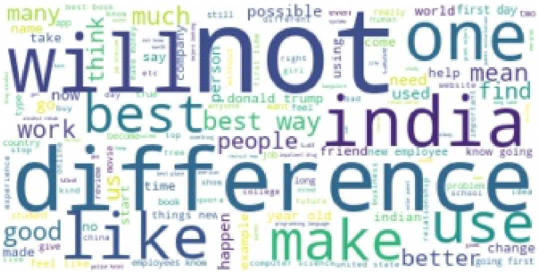
rs 1k
make money
hillary clinton
earn money
india best
good
not
best
way
ask question
improve english
book
best book
money online
black money
best place
new year
donald trump
questions
quora
people ask
rupee note
1k rupee
1k note
lose weight
difference
world war
programming language

Word Clouds generated from non duplicate pair question's text

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:
```

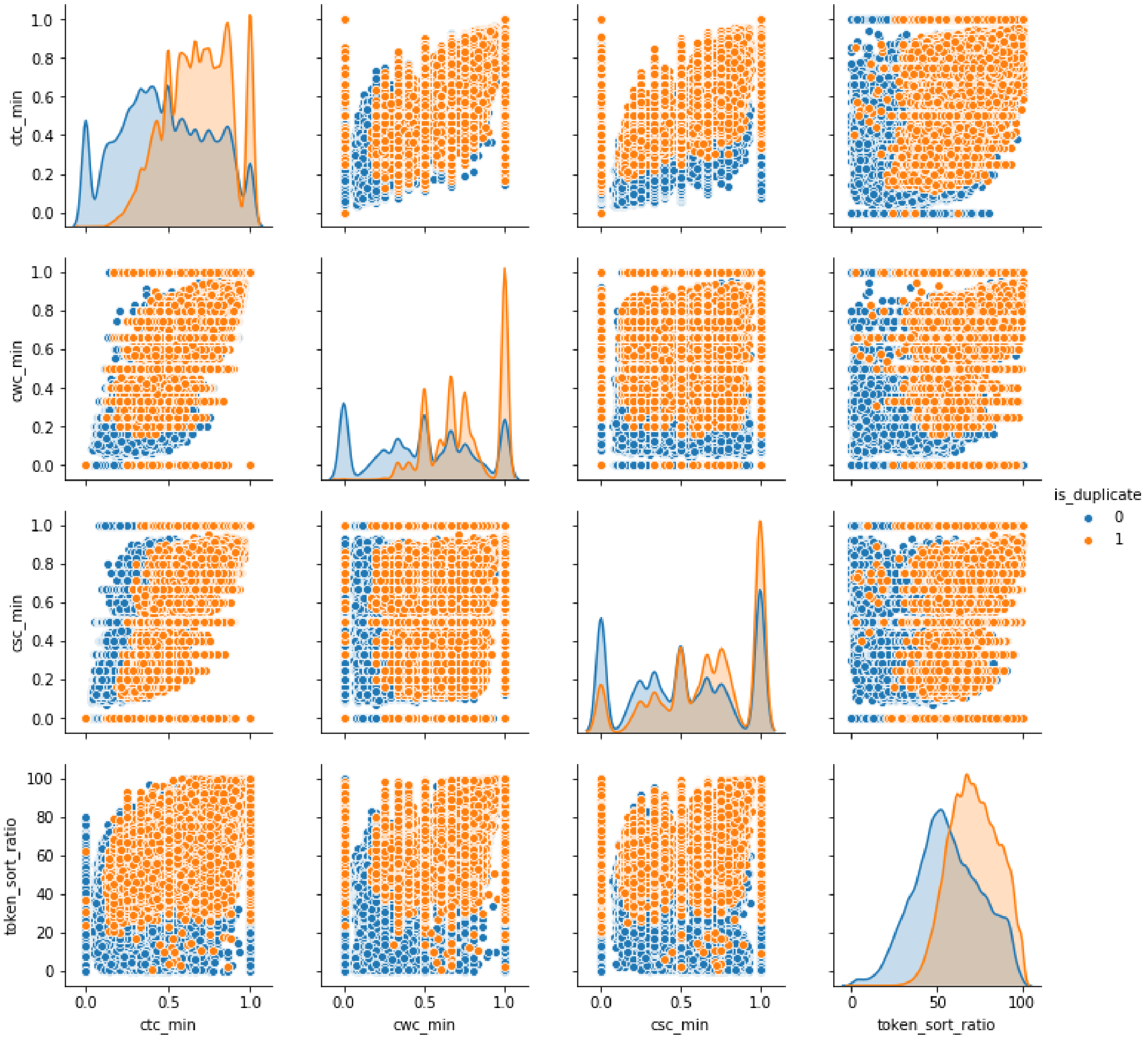


3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

In [28]:

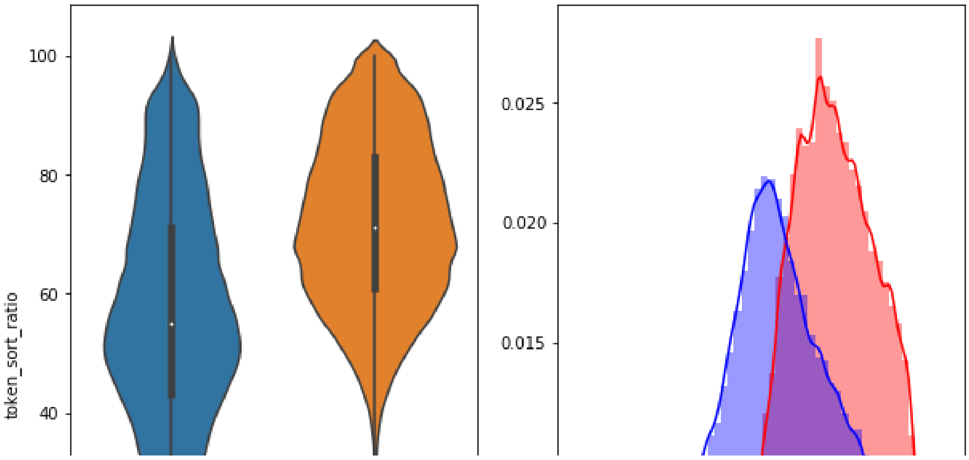
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_du
plicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
```

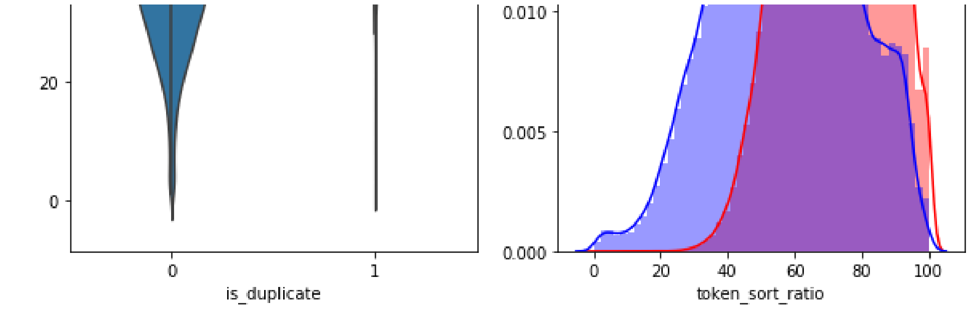
```
plt.show()
```



In [29]:

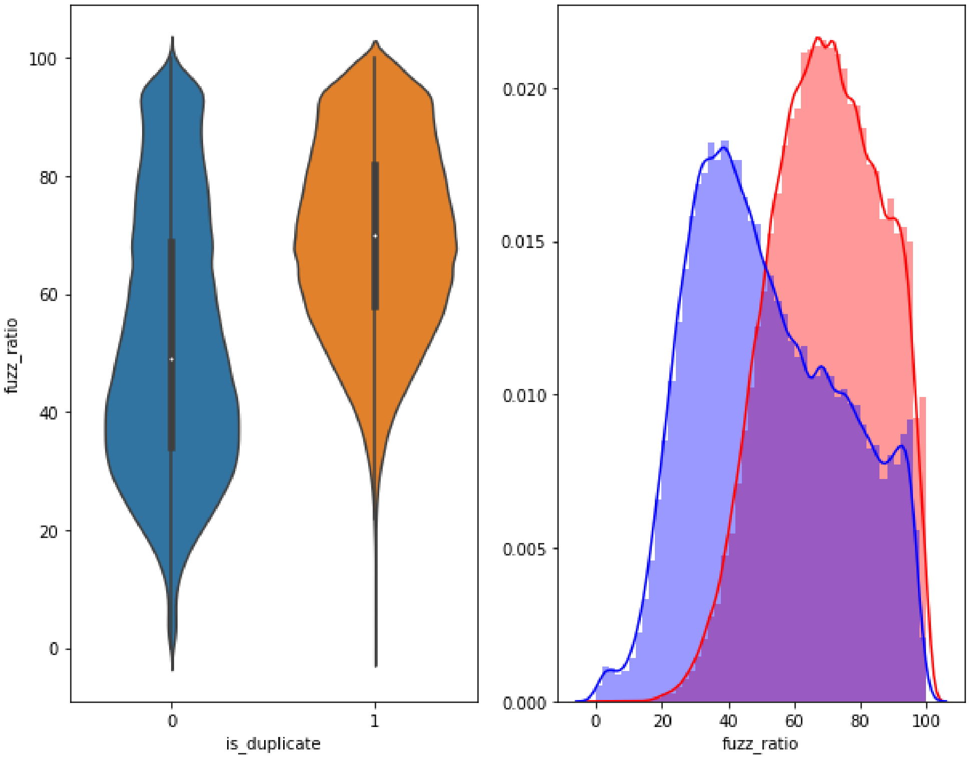
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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()
```



3.5.2 Visualization

In [31]:

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

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' , 'toke
n_sort_ratio' ,  'fuzz_ratio' , 'fuzz_partial_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323: D
ataConversionWarning:
```

```
Data with input dtype int64, float64 were all converted to float64 by MinMaxScaler.
```

In [32]:

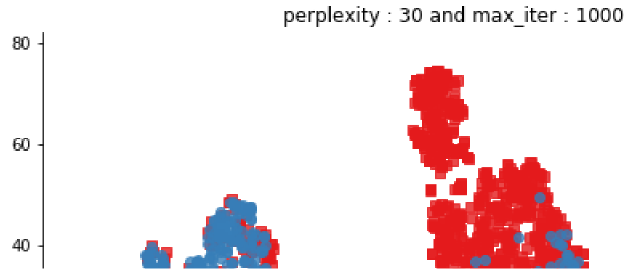
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

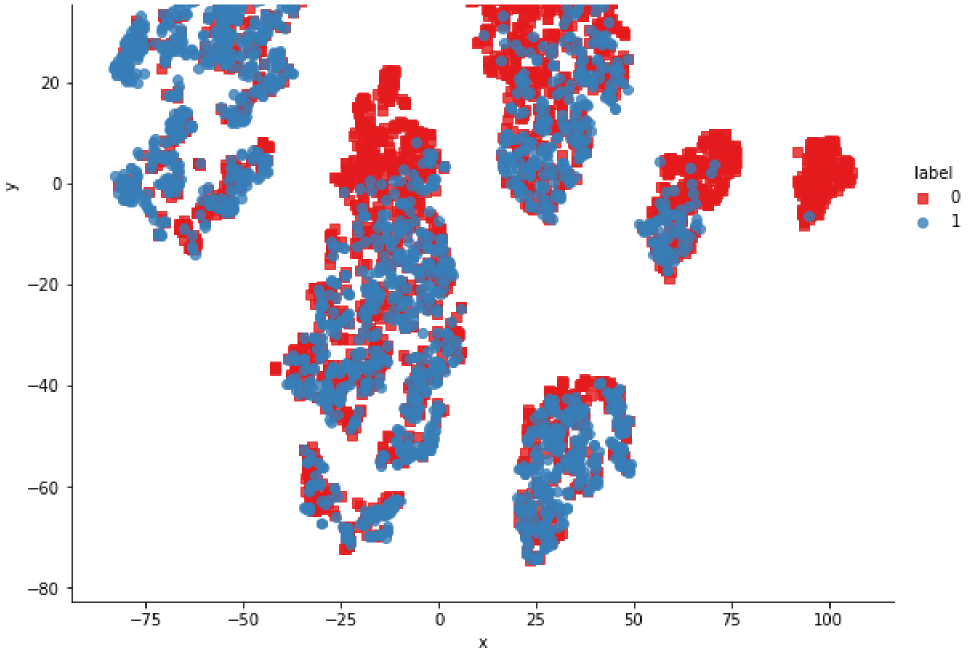
```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.024s...
[t-SNE] Computed neighbors for 5000 samples in 0.570s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional 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: 0.130446
[t-SNE] Computed conditional probabilities in 0.263s
[t-SNE] Iteration 50: error = 81.2911148, gradient norm = 0.0457501 (50 iterations in 4.173s)
[t-SNE] Iteration 100: error = 70.6044159, gradient norm = 0.0086692 (50 iterations in 2.808s)
[t-SNE] Iteration 150: error = 68.9124908, gradient norm = 0.0056016 (50 iterations in 2.441s)
[t-SNE] Iteration 200: error = 68.1010742, gradient norm = 0.0047585 (50 iterations in 2.356s)
[t-SNE] Iteration 250: error = 67.5907974, gradient norm = 0.0033576 (50 iterations in 2.592s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.590797
[t-SNE] Iteration 300: error = 1.7929677, gradient norm = 0.0011899 (50 iterations in 2.631s)
[t-SNE] Iteration 350: error = 1.3937442, gradient norm = 0.0004817 (50 iterations in 2.701s)
[t-SNE] Iteration 400: error = 1.2280033, gradient norm = 0.0002773 (50 iterations in 2.344s)
[t-SNE] Iteration 450: error = 1.1383208, gradient norm = 0.0001865 (50 iterations in 2.502s)
[t-SNE] Iteration 500: error = 1.0834006, gradient norm = 0.0001423 (50 iterations in 2.625s)
[t-SNE] Iteration 550: error = 1.0474092, gradient norm = 0.0001144 (50 iterations in 2.920s)
[t-SNE] Iteration 600: error = 1.0231259, gradient norm = 0.0000995 (50 iterations in 2.725s)
[t-SNE] Iteration 650: error = 1.0066353, gradient norm = 0.0000895 (50 iterations in 2.704s)
[t-SNE] Iteration 700: error = 0.9954656, gradient norm = 0.0000805 (50 iterations in 2.707s)
[t-SNE] Iteration 750: error = 0.9871529, gradient norm = 0.0000719 (50 iterations in 2.504s)
[t-SNE] Iteration 800: error = 0.9801921, gradient norm = 0.0000657 (50 iterations in 2.716s)
[t-SNE] Iteration 850: error = 0.9743395, gradient norm = 0.0000631 (50 iterations in 2.950s)
[t-SNE] Iteration 900: error = 0.9693972, gradient norm = 0.0000606 (50 iterations in 2.876s)
[t-SNE] Iteration 950: error = 0.9654404, gradient norm = 0.0000594 (50 iterations in 2.519s)
[t-SNE] Iteration 1000: error = 0.9622302, gradient norm = 0.0000565 (50 iterations in 2.737s)
[t-SNE] KL divergence after 1000 iterations: 0.962230
```

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()
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\seaborn\regression.py:546: UserWarni
ng:

The `size` paramter has been renamed to `height`; please update your code.
```





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 0.023s...
[t-SNE] Computed neighbors for 5000 samples in 0.569s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.280s
[t-SNE] Iteration 50: error = 80.5316772, gradient norm = 0.0296611 (50 iterations in 12.988s)
[t-SNE] Iteration 100: error = 69.3815765, gradient norm = 0.0033166 (50 iterations in 6.924s)
[t-SNE] Iteration 150: error = 67.9724655, gradient norm = 0.0018542 (50 iterations in 5.745s)
[t-SNE] Iteration 200: error = 67.4176865, gradient norm = 0.0012513 (50 iterations in 5.745s)
[t-SNE] Iteration 250: error = 67.1036377, gradient norm = 0.0009096 (50 iterations in 6.026s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.103638
[t-SNE] Iteration 300: error = 1.5251231, gradient norm = 0.0007399 (50 iterations in 7.823s)
[t-SNE] Iteration 350: error = 1.1820215, gradient norm = 0.0002076 (50 iterations in 9.283s)
[t-SNE] Iteration 400: error = 1.0389463, gradient norm = 0.0000969 (50 iterations in 8.707s)
[t-SNE] Iteration 450: error = 0.9659566, gradient norm = 0.0000635 (50 iterations in 9.077s)
[t-SNE] Iteration 500: error = 0.9267892, gradient norm = 0.0000482 (50 iterations in 8.771s)
[t-SNE] Iteration 550: error = 0.9053178, gradient norm = 0.0000406 (50 iterations in 9.042s)
[t-SNE] Iteration 600: error = 0.8915660, gradient norm = 0.0000349 (50 iterations in 9.348s)
[t-SNE] Iteration 650: error = 0.8804696, gradient norm = 0.0000345 (50 iterations in 8.849s)
[t-SNE] Iteration 700: error = 0.8723292, gradient norm = 0.0000358 (50 iterations in 8.926s)
[t-SNE] Iteration 750: error = 0.8668707, gradient norm = 0.0000314 (50 iterations in 9.174s)
[t-SNE] Iteration 800: error = 0.8626194, gradient norm = 0.0000250 (50 iterations in 8.840s)
[t-SNE] Iteration 850: error = 0.8584315, gradient norm = 0.0000253 (50 iterations in 9.398s)
[t-SNE] Iteration 900: error = 0.8547347, gradient norm = 0.0000261 (50 iterations in 9.195s)
[t-SNE] Iteration 950: error = 0.8517873, gradient norm = 0.0000263 (50 iterations in 8.738s)
[t-SNE] Iteration 1000: error = 0.8493521, gradient norm = 0.0000250 (50 iterations in 9.073s)
[t-SNE] KL divergence after 1000 iterations: 0.849352
```

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')
```

4.3 Random train test split(70:30)

In [2]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('nlp_features_train.csv'):
```

```
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")


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

In [3]:

```
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','is_duplicate'],axis=1)

df  = df1.merge(df2, on='id',how='left')
df  = df.merge(df3, on='id',how='left')
```

In [4]:

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

In [5]:

```
df.head(5)
```

Out[5]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	...	q2len	q1_n_words
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	...	57	14
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	...	88	8
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	...	59	14
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	...	65	11
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	...	39	13

5 rows × 30 columns

In [6]:

```
df.shape
```

Out[6]:

```
(100000, 30)
```

In [7]:

```
X_train,X_test, y_train, y_test = train_test_split(df.drop(['is_duplicate'],axis=1), df['is_duplicate']
, stratify=df['is_duplicate'], test_size=0.3)
```

In [8]:

```
X_train.shape
```

Out[8]:

```
(70000, 29)
```

In [9]:

```
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 : (70000, 29)
Number of data points in test data : (30000, 29)
```

In [10]:

```
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:  0.6274571428571428 Class 1:  0.3725428571428571
---------- Distribution of output variable in train data ----------
Class 0:  0.3725333333333333 Class 1:  0.3725333333333333
```

3.6 Featurizing text data with tfidf weighted word-vectors

In [11]:

```
# 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 ---------------------
X_train['question1'] = X_train['question1'].apply(lambda x: str(x))
X_train['question2'] = X_train['question2'].apply(lambda x: str(x))
X_test['question1'] = X_test['question1'].apply(lambda x: str(x))
X_test['question2'] = X_test['question2'].apply(lambda x: str(x))
```

In [12]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
#from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions_X_train = X_train['question1'] + " " + X_train['question2']
questions_X_test = X_test['question1'] + " " + X_test['question2']

tfidf = TfidfVectorizer(lowercase=False, )
tfidf_vect = tfidf.fit(questions_X_train)
tfidf_vect = tfidf.fit(questions_X_train)
tfidf_vect_X_train = tfidf_vect.transform(questions_X_train)
tfidf_vect_X_test = tfidf_vect.transform(questions_X_test)

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

In [13]:

```
tfidf_vect_X_train.shape
```

Out[13]:

```
(70000, 47679)
```

In [14]:

```
#function to get tfidf word2Vec for sentances
def tfidfw2v(data,w2v_vocab,w2v_model,tf_idf_vect, dictionary):
    tfidf_sent_vectors = []
    row=0;
    tfidf_feat = tf_idf_vect.get_feature_names()
    #prepare the corresponding vectors for sentances
    for review in tqdm(data):
        review_vector = np.zeros(50)
        weight_sum =0
        for w in review.split(" "):
            if (w in w2v_vocab) and (w in tfidf_feat):
                word_vector = w2v_model.wv[w]
                tf_idf = dictionary[w]*(review.count(w)/len(review))
                review_vector += (word_vector*tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            review_vector /= weight_sum
        tfidf_sent_vectors.append(review_vector)
        row += 1

    return tfidf_sent_vectors
```

In [15]:

```
from gensim.models import Word2Vec

#training W2V on training data
list_of_sent=[]
#we are using the training data that we splitted as per time based split
for sent in questions_X_train:
    list_of_sent.append(sent.split())

#preparing our own Word2Vec model
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)

#get all the words in word2Vec
w2v_words = list(w2v_model.wv.vocab)
```

```
C:\Users\bolua\AppData\Local\Continuum\anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning:

detected Windows; aliasing chunkize to chunkize_serial
```

In [16]:

```
# We have pre stored w2v data, if we are running prog for first time and we dont have data the below code will be executed
if not os.path.isfile('train_with_w2v.csv'):
    dictionary = dict(zip(tfidf_vect.get_feature_names(), list(tfidf_vect.idf_)))

    tfidf_w2v_traindata = tfidfw2v(questions_X_train,w2v_words,w2v_model,tfidf_vect,dictionary)
    tfidf_w2v_testdata = tfidfw2v(questions_X_test,w2v_words,w2v_model,tfidf_vect,dictionary)
```

```
100%|██████████████████████████████████████████████████| 70000/70000 [38:37<0
0:00, 30.21it/s]
100%|██████████████████████████████████████████████████████████| 30000/30000 [15:55<0
0:00, 30.40it/s]
```

train W2v on trained data get TFIDF w2v for trained and test data both store the TFIDFW2V values some different variable Use appropritly as per the models Use for table to differentiate the performances

In [56]:

```
#df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
#df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
#df3 = df1.merge(df2, on='id',how='left')
#df3_q = pd.DataFrame(df3.q_feats_m.values.tolist(), index= df3.index)

#df = df.drop(['question1','question2','is_duplicate'],axis=1)
X_train = X_train.drop(['question1','question2'],axis=1)
X_test = X_test.drop(['question1','question2'],axis=1)


X_train_tfidf = hstack((X_train, tfidf_vect_X_train))
X_test_tfidf = hstack((X_test, tfidf_vect_X_test))
```

In [77]:

```
#We will perform below operation only when we dont have saved W2v data train and test wise
if not os.path.isfile('train_with_w2v.csv'):
    df_w2v_Xtrain = X_train #Copying the train data don't want to mesh with the existing data
    df_w2v_Xtest = X_test

    X_train.head(5)
```

Out[77]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	...	q1len	q2l
12277	12277	0.666644	0.399992	0.499975	0.249994	0.499992	0.333330	0.0	0.0	3.0	...	35	
70069	70069	0.999975	0.666656	0.499995	0.454541	0.642853	0.529409	1.0	0.0	3.0	...	73	
50332	50332	0.399992	0.399992	0.624992	0.555549	0.499996	0.437497	1.0	1.0	2.0	...	71	
97149	97149	0.999967	0.999967	0.249994	0.199996	0.571420	0.499994	1.0	1.0	1.0	...	31	
74235	74235	0.499992	0.428565	0.285710	0.249997	0.384612	0.312498	0.0	0.0	3.0	...	81	

5 rows × 28 columns

In [78]:

```
#We will perform below operation only when we dont have saved W2v data train and test wise
if not os.path.isfile('train_with_w2v.csv'):
    df_w2v_Xtrain['qtfidfw2v'] = list(tfidf_w2v_traindata) #Get the W2v data and store it in to a colum
n, all data in one column
    df_w2v_Xtest['qtfidfw2v'] = list(tfidf_w2v_testdata)

    train_w2v = pd.DataFrame(df_w2v_Xtrain.qtfidfw2v.values.tolist(), index= df_w2v_Xtrain.index) # it
will give a w2v data column wise, single column single value
    test_w2v = pd.DataFrame(df_w2v_Xtest.qtfidfw2v.values.tolist(), index= df_w2v_Xtest.index)
    train_w2v.head()
```

Out[78]:

0 1 2 3 4 5 6 7 8 9 10 11

	0	1	2	3	4	5	6	7	8	9	...	40	41
12277	0.142405	0.056168	-0.548278	-0.605494	0.834295	-0.484620	-0.089185	1.077515	0.038885	-0.110704	...	0.988204	0.615726
70069	0.262226	-0.069784	-1.273017	0.350692	0.267639	-0.287341	0.666738	0.695265	-0.043503	1.112599	...	-0.572716	0.611933
50332	-1.490995	-0.761525	0.105540	0.496346	0.107924	-1.022299	0.000582	-0.468595	0.023732	-0.043532	...	0.164122	0.078225
97149	-0.480048	1.841199	-2.008336	1.100040	0.499918	-0.728353	1.972168	1.400581	1.569053	0.116722	...	-0.743827	0.225178
74235	-0.437450	-0.531680	0.289087	0.519973	1.168459	-1.197193	-0.014980	0.073620	-0.339947	0.622424	...	0.003728	0.347412

5 rows × 50 columns

In [104]:

```
X_train_tfidfw2v = pd.read_csv("train_with_w2v.csv")
X_test_tfidfw2v = pd.read_csv('test_with_w2v.csv')
```

In [105]:

```
X_train_tfidfw2v.head()
```

Out[105]:

	Unnamed: 0	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	...	40	
0	0	12277	0.666644	0.399992	0.499975	0.249994	0.499992	0.333330	0.0	0.0	...	0.988204	0.6157
1	1	70069	0.999975	0.666656	0.499995	0.454541	0.642853	0.529409	1.0	0.0	...	-0.572716	0.6119
2	2	50332	0.399992	0.399992	0.624992	0.555549	0.499996	0.437497	1.0	1.0	...	0.164122	0.0782
3	3	97149	0.999967	0.999967	0.249994	0.199996	0.571420	0.499994	1.0	1.0	...	-0.743827	0.2251
4	4	74235	0.499992	0.428565	0.285710	0.249997	0.384612	0.312498	0.0	0.0	...	0.003728	0.3474

5 rows × 79 columns

In [18]:

```
###################################################################################################
#########################
```

In [19]:

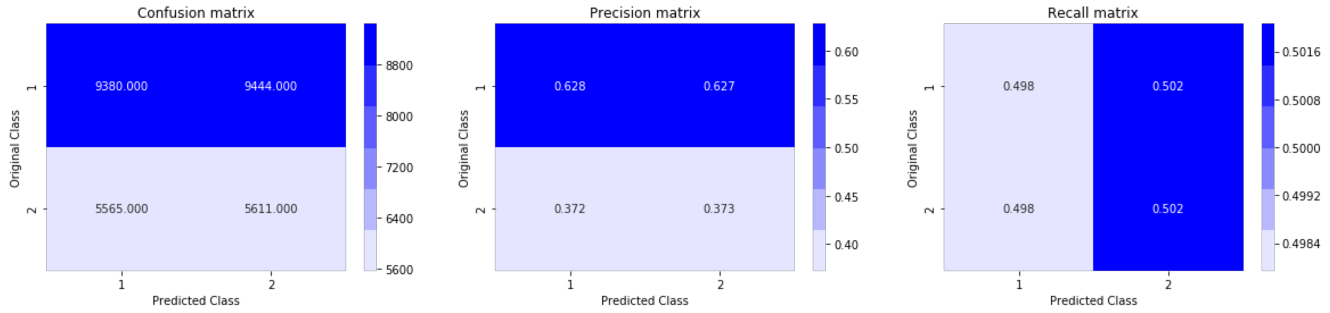
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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]]
    #                              [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 [128]:

```
# 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])

test_log_loss = log_loss(y_test, predicted_y, eps=1e-15)
print("Log loss on Test Data using Random Model",test_log_loss)

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

```
Log loss on Test Data using Random Model 0.88964757781306
```



In [126]:

In [126]:

```
from prettytable import PrettyTable

out_table = PrettyTable()
#x.del_row(1)
out_table.field_names = ["Model", "Vectrozier","Hypar parameter","train log loss","test log loss"]
```

In [129]:

```
out_table.add_row(["Random","NA","NA","NA",test_log_loss])
print(out_table)
```

```
+--------+------------+--------------------------------------------+--------------------+--------------
------+
| Model  | Vectrozier |              Hypar parameter               |   train log loss   |   test log lo
ss    |
+--------+------------+--------------------------------------------+--------------------+--------------
------+
|  GBDT  |  tfidfw2v  | learning_rate = 0.1 and n_estimators = 100 | 0.3557517596257164 | 0.35856407182
36129 |
| Random |     NA     |                     NA                     |         NA         |  0.8896475778
1306  |
+--------+------------+--------------------------------------------+--------------------+--------------
------+
```

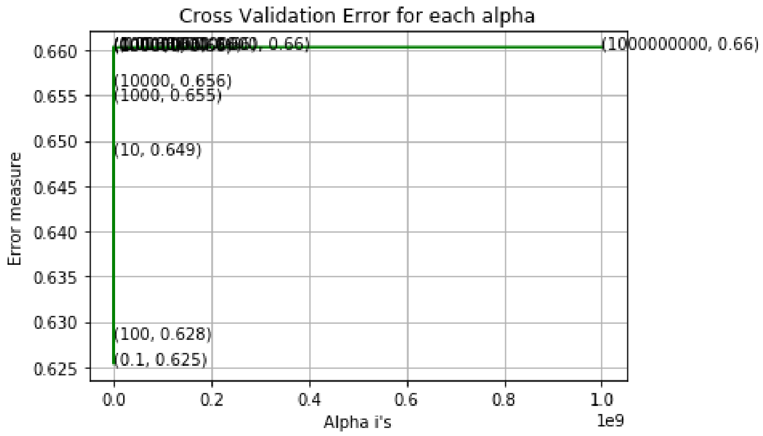
4.4 Logistic Regression with hyperparameter tuning

In [131]:

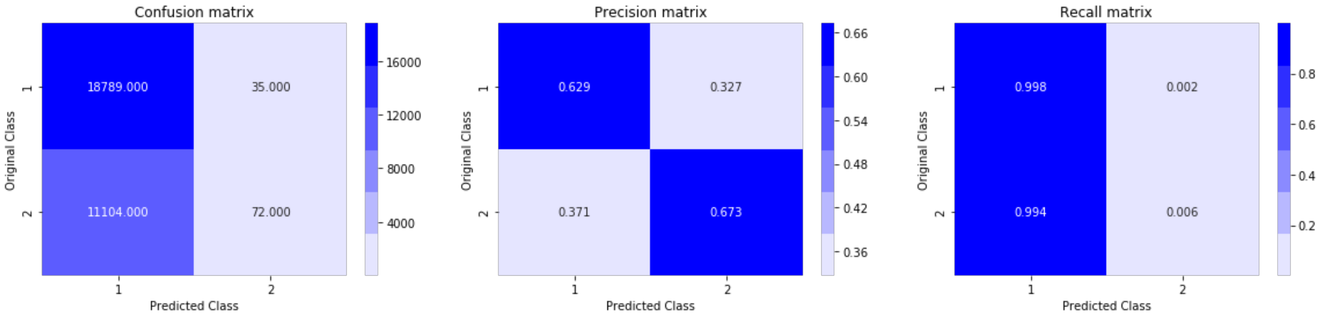
```
alpha = [10 ** x for x in range(-2, 10)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, 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',class_weight='balanced', random_state=42)
    clf.fit(X_train_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    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.classe
s_, eps=1e-15))

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

```
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',class_weight='balanced', random_s
tate=42)
clf.fit(X_train_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_tfidf, y_train)

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

predict_y = sig_clf.predict_proba(X_test_tfidf)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_log_loss)

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 =  0.01 The log loss is: 0.6602902109101162
For values of alpha =  0.1 The log loss is: 0.6253418034549929
For values of alpha =  1 The log loss is: 0.6602902109101162
For values of alpha =  10 The log loss is: 0.6485515202393919
For values of alpha =  100 The log loss is: 0.6282362083998229
For values of alpha =  1000 The log loss is: 0.6545326238142921
For values of alpha =  10000 The log loss is: 0.6562109984313237
For values of alpha =  100000 The log loss is: 0.6600618466003969
For values of alpha =  1000000 The log loss is: 0.6601735021472678
For values of alpha =  10000000 The log loss is: 0.660276025287811
For values of alpha =  100000000 The log loss is: 0.6603064395790738
For values of alpha =  1000000000 The log loss is: 0.6603036680055261
```



```
For values of best alpha =  0.1 The train log loss is: 0.6264551646455128
For values of best alpha =  0.1 The test log loss is: 0.6253418034549929
Total number of data points : 30000
```



In [132]:

```
out_table.add_row(["LR","tfidf","alpha = {0}".format(best_alpha),train_log_loss,test_log_loss])
print(out_table)
```

```
+---------+-------------+----------------------------------------------+--------------------+--------------
```

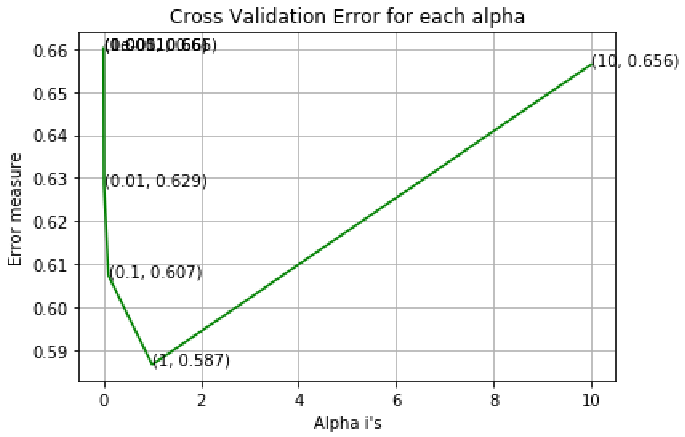
```
------+
| Model  | Vectrozier |               Hypar parameter               |   train log loss   |   test log lo
ss     |
+--------+------------+---------------------------------------------+--------------------+--------------
------+
|  GBDT  |  tfidfw2v  | learning_rate = 0.1 and n_estimators = 100 | 0.3557517596257164 | 0.35856407182
36129 |
| Random |     NA     |                      NA                     |         NA         |  0.8896475778
1306  |
|   LR   |   tfidf    |                  alpha = 1                  | 0.6264551646455128 | 0.62534180345
49929 |
+--------+------------+---------------------------------------------+--------------------+--------------
------+
```

4.5 Linear SVM with hyperparameter tuning

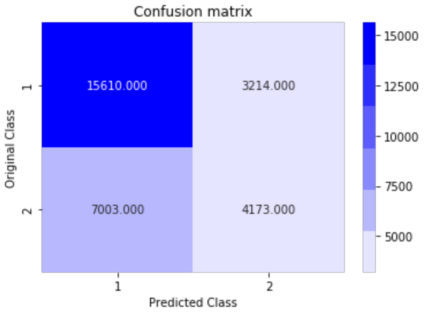
In [133]:

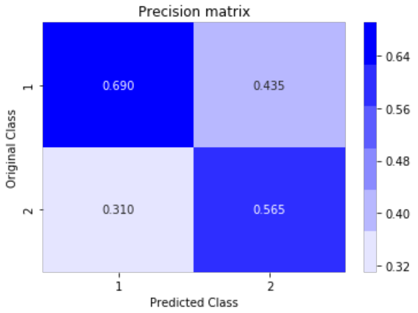
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, 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',class_weight='balanced', random_state=42)
    clf.fit(X_train_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    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.classe
s_, 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',class_weight='balanced', random
_state=42)
clf.fit(X_train_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_tfidf, y_train)

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

predict_y = sig_clf.predict_proba(X_test_tfidf)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_log_loss)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_log_loss)

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

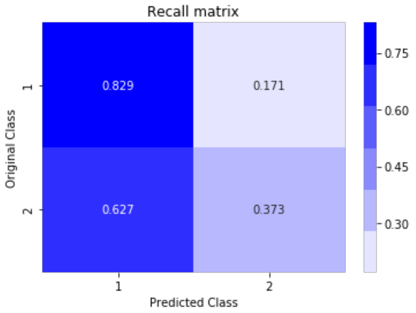
```
For values of alpha =  1e-05 The log loss is: 0.6602902109101162
For values of alpha =  0.0001 The log loss is: 0.6602902109101162
For values of alpha =  0.001 The log loss is: 0.6602902109101162
For values of alpha =  0.01 The log loss is: 0.6285456172629476
For values of alpha =  0.1 The log loss is: 0.6073679162200024
For values of alpha =  1 The log loss is: 0.5865827096310336
For values of alpha =  10 The log loss is: 0.6564304236252156
```



```
For values of best alpha =  1 The train log loss is: 0.5889182927200233
For values of best alpha =  1 The test log loss is: 0.5865827096310336
Total number of data points : 30000
```







```
In [134]:
```

```
out_table.add_row(["Linear SVM","tfidf","alpha = {0}".format(best_alpha),train_log_loss,test_log_loss])
print(out_table)
```

```
+------------+------------+--------------------------------------------+--------------------+----------
----------+
|   Model    | Vectrozier |              Hypar parameter               |   train log loss   |   test lo
g loss     |
+------------+------------+--------------------------------------------+--------------------+----------
----------+
|    GBDT    |  tfidfw2v  | learning_rate = 0.1 and n_estimators = 100 | 0.3557517596257164 | 0.3585640
718236129 |
|   Random   |     NA     |                     NA                     |         NA         |  0.889647
57781306  |
|     LR     |   tfidf    |                 alpha = 1                  | 0.6264551646455128 | 0.6253418
034549929 |
| Linear SVM |   tfidf    |                 alpha = 5                  | 0.5889182927200233 | 0.5865827
096310336 |
+------------+------------+--------------------------------------------+--------------------+----------
----------+
```

4.6 XGBoost

4.0 XGBoost

In [85]:

```
import math
import collections
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

def plot_train_test_acc(cv_results_):
    X2_Val=cv_results_['param_learning_rate ']
    X1_Val=cv_results_['param_n_estimators ']
    Y1=[list(i) for i in dict(zip(X1_Val,cv_results_['mean_test_score'])).items()]
    Y1 = sorted(Y1)
    Y2=[list(i) for i in dict(zip(X1_Val,cv_results_['mean_train_score'])).items()]
    Y2 = sorted(Y2)

    Z1=[list(i) for i in dict(zip(X2_Val,cv_results_['mean_test_score'])).items()]
    Z1 = sorted(Z1)
    Z2=[list(i) for i in dict(zip(X2_Val,cv_results_['mean_train_score'])).items()]
    Z2 = sorted(Z2)

    y1 = [math.log(X[0]) for X in Z1]
    x1 = [math.log(X[0]) for X in Y1]
    z1 = [X[1] for X in Y1]



    y2 = [math.log(X[0]) for X in Z1]
    x2 = [math.log(X[0]) for X in Y1]
    z2 = [X[1] for X in Y2]

    # https://plot.ly/python/3d-axes/
    trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Cross validation')
    trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'train')
    data = [trace1, trace2]

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [107]:

```
# 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(X_train_tfidfw2v.columns)
for i in cols:
    X_train_tfidfw2v[i] = X_train_tfidfw2v[i].apply(pd.to_numeric,errors='coerce')
    print(i)
```

```
Unnamed: 0
id
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
```

```
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
qtfidfw2v
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

In [108]:

```
X_test_tfidfw2v.head()
```

Out[108]:

	Unnamed: 0	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	...	40	
0	0	20532	0.000000	0.000000	0.999950	0.666644	0.666644	0.333328	0.0	1.0	...	0.753673	3.0341
1	1	96820	0.999950	0.999950	0.999900	0.499975	0.999967	0.749981	1.0	0.0	...	1.656421	2.1242

	Unnamed: 0	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	...	40	
2	2	83896	0.833319	0.555549	0.749981	0.428565	0.615380	0.571424	1.0	0.0	...	0.745544	0.5458
3	3	45128	0.499988	0.399992	0.666644	0.249997	0.499994	0.307690	0.0	1.0	...	0.281154	0.3250
4	4	52292	0.833319	0.714276	0.999989	0.999989	0.636361	0.608693	0.0	1.0	...	1.192994	0.2856

5 rows × 79 columns

In [109]:

```
# 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(X_test_tfidfw2v.columns)
for i in cols:
    X_test_tfidfw2v[i] = X_test_tfidfw2v[i].apply(pd.to_numeric,errors='coerce')
    print(i)
```

```
Unnamed: 0
id
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
qtfidfw2v
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

In [110]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

clf = xgb.XGBClassifier(subsample = 0.6)
tuned_parameters = {'learning_rate ': [0.001,0.01,0.01,0.1],'n_estimators ': [50,100,150,200]}
#alpha_range=uniform(10**-4,10**4)
#tuned_parameters = {'alpha': alpha_range}
model = RandomizedSearchCV(clf,tuned_parameters,scoring="roc_auc", cv=5)
model.fit(X_train_tfidfw2v, y_train)
#getting y values using our trained model
pred_cv = model.score(X_test_tfidfw2v, y_test)
pred_train = model.score(X_train_tfidfw2v, y_train)
best_estimate = model.best_estimator_
cv_results_ = model.cv_results_
```

In [111]:

```
print(best_estimate)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, learning_rate =0.01,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, 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=True,
       subsample=0.6)
```

In [120]:

```
#plot_train_test_acc(cv_results_)

predict_y = best_estimate.predict_proba(X_train_tfidfw2v)
train_log_loss = log_loss(y_train, predict_y)
print('For values of learning_rate = {0} and n_estimators = {1} is'.format(best_estimate.learning_rate,
best_estimate.n_estimators),  "The train log loss is:",train_log_loss)
```

```
For values of learning_rate = 0.1 and n_estimators = 100 is The train log loss is: 0.3557517596257164
```

In [121]:

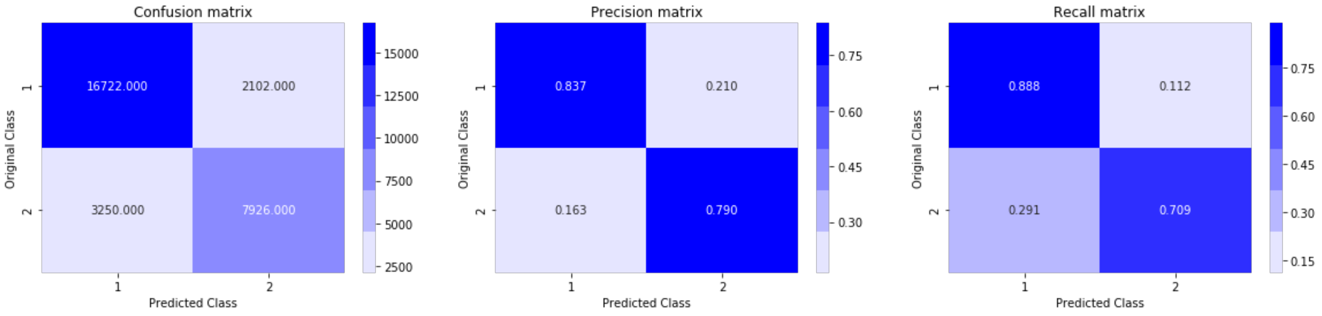
```
predict_y = best_estimate.predict_proba(X_test_tfidfw2v)
test_log_loss = log_loss(y_test, predict_y, eps=1e-15)
print('For values of learning_rate = {0} and n_estimators = {1} is'.format(best_estimate.learning_rate,
best_estimate.n_estimators), "The test log loss is:",test_log_loss)
```

```
For values of learning_rate = 0.1 and n_estimators = 100 is The test log loss is: 0.3585640718236129
```

In [124]:

```
predict_y = best_estimate.predict(X_test_tfidfw2v)
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
```



In [135]:

```
out_table.add_row(["GBDT","tfidfw2v","learning_rate = {0} and n_estimators = {1}".format(best_estimate.
learning_rate,best_estimate.n_estimators),train_log_loss,test_log_loss])
print(out_table)
```

```
+------------+------------+-------------------------------------------+--------------------+----------
----------+
|   Model    | Vectrozier |              Hypar parameter              |   train log loss   |   test lo
g loss    |
+------------+------------+-------------------------------------------+--------------------+----------
----------+
|    GBDT    |  tfidfw2v  | learning_rate = 0.1 and n_estimators = 100 | 0.3557517596257164 | 0.3585640
718236129 |
|   Random   |     NA     |                     NA                    |         NA         |  0.889647
57781306  |
|     LR     |   tfidf    |                 alpha = 1                 | 0.6264551646455128 | 0.6253418
034549929 |
| Linear SVM |   tfidf    |                 alpha = 5                 | 0.5889182927200233 | 0.5865827
096310336 |
+------------+------------+-------------------------------------------+--------------------+----------
----------+
```

Conclusions

From the above plots it looks like XG boost is performing very good Assumption behind using tfidf for LR and Linear SVM not helping much, the results are not up to mark but not worst also. They are performing better then random mondel