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

__ Useful Links __

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments (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 (https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0)
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning (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 (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 (https://www.kaggle.com/c/quora-question-pairs#evaluation)

Metric(s):

- log-loss : https://www.kaggle.com/wiki/LogarithmicLoss (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

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

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:

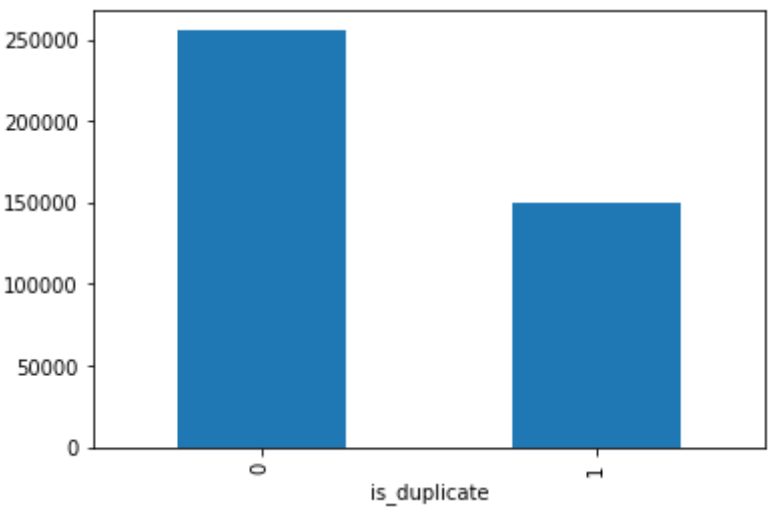
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

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 0x1f886555240>
```



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,
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_duplicate'].mean()*100, 2)))
```

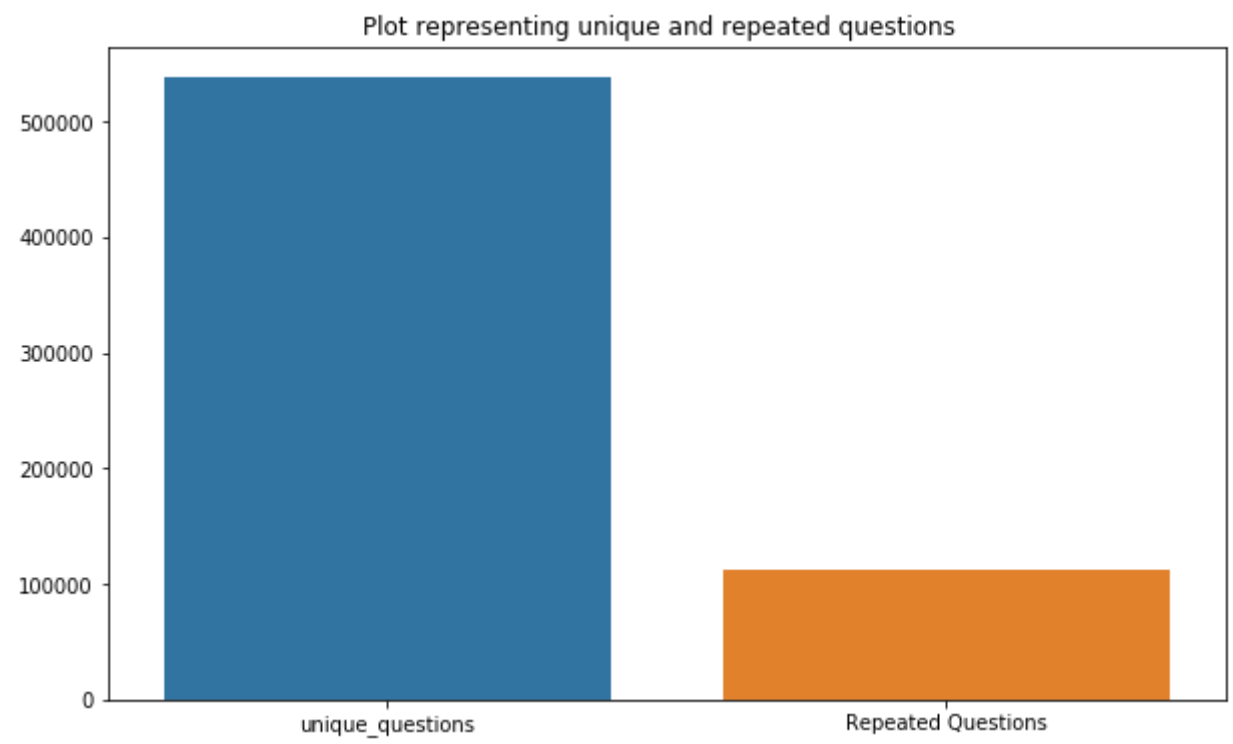
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

~> 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_on

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

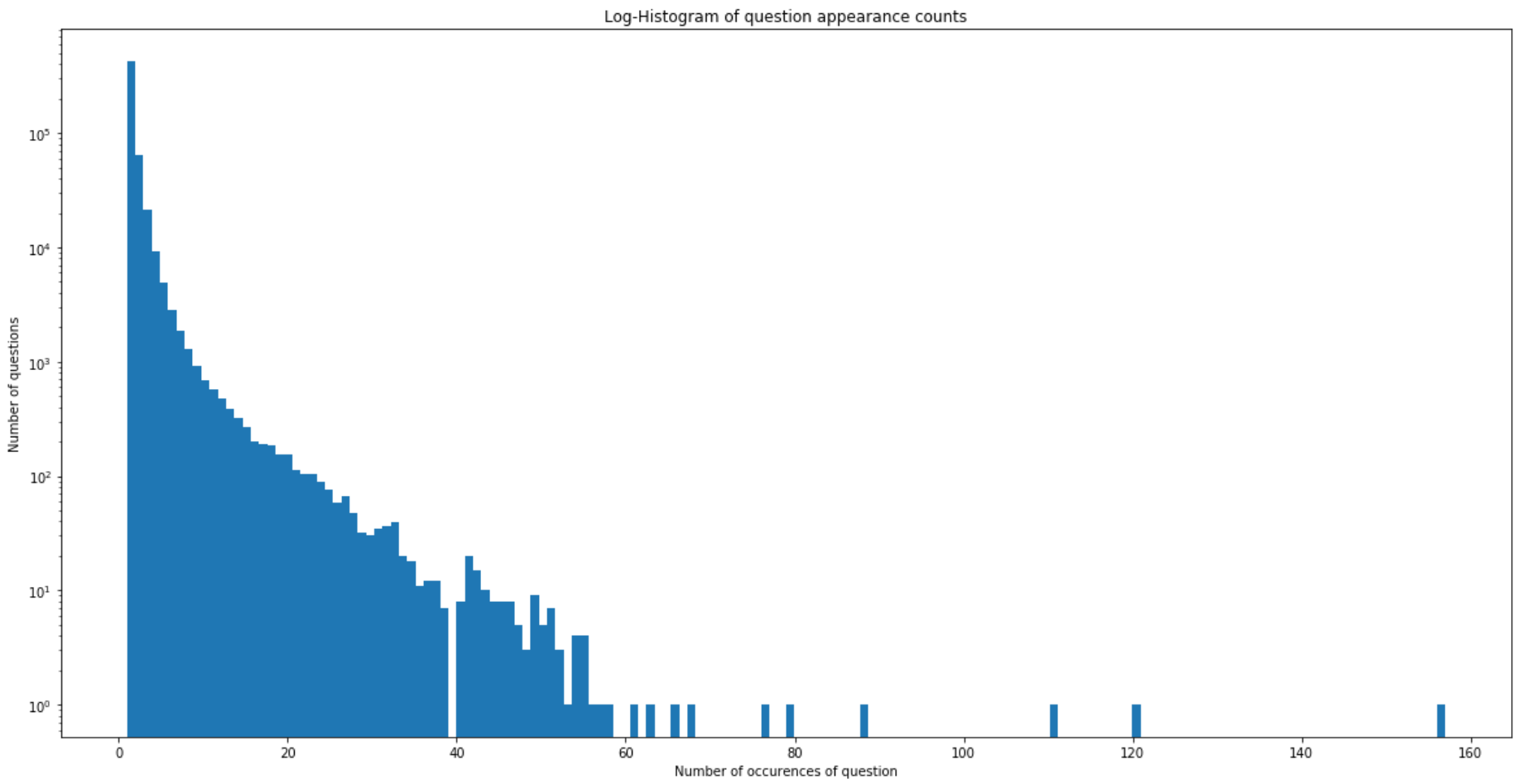
In [10]:
```
#checking whether there are any repeated pair of questions

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]:
```
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_Common	word_Total	w
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	23.0	
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	20.0	
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	24.0	
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	19.0	
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	20.0	

3.3.1 Analysis of some of the extracted features

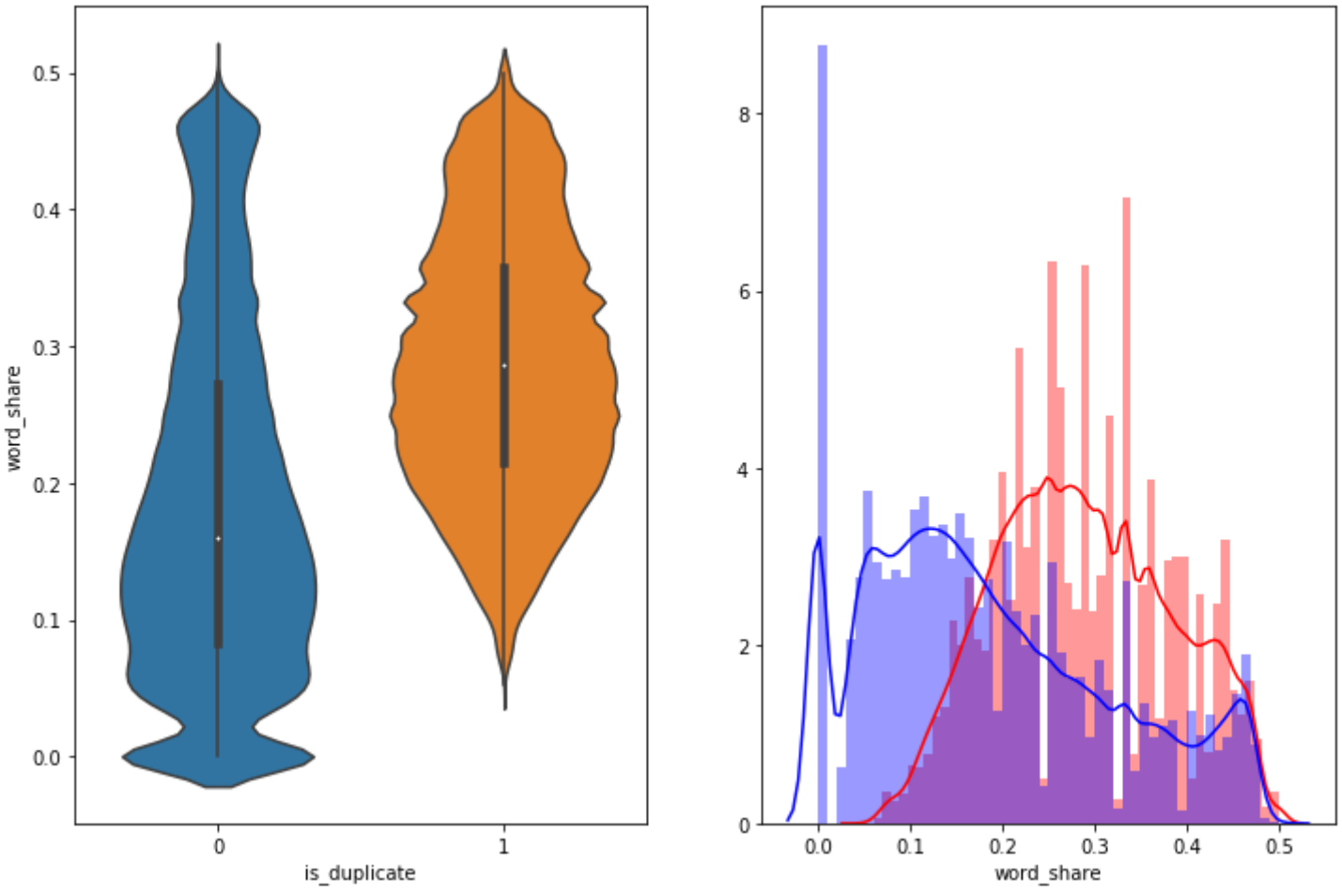
- Here are some questions have only one single words.

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

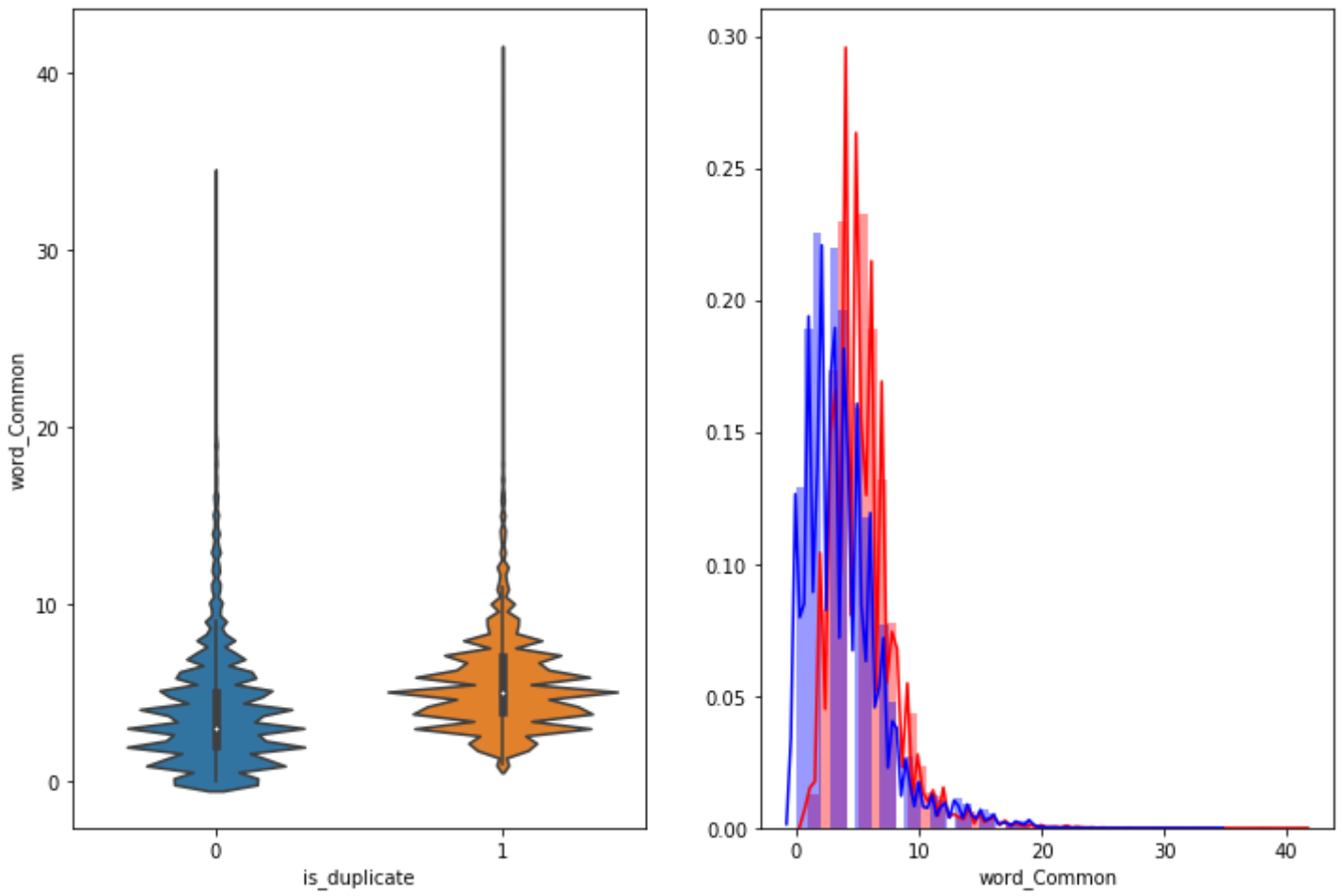
In [16]:
```
plt.figure(figsize=(12, 8))

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]:
```
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]:
```
#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 [19]:
```
df.head(2)
```

Out[19]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	wo
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	23.0	
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	20.0	

3.4 Preprocessing of Text

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

In [20]:
```
# 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 = 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 (https://github.com/seatgeek/fuzzywuzzy#usage)
  http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

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

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

- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage)
  http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (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 [21]:
```
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 alphabetically, 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["question2"]), axis=1)
    return df
```

In [22]:
```
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[22]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	first_word_eq	abs_len_d
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	1.0	2
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	1.0	5

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 [23]:
```
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',encoding='utf8')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s',encoding='utf8')
```

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

In [24]:
```
# 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 : 16110303
Total number of words in non duplicate pair questions : 33194892
```

__ Word Clouds generated from duplicate pair question's text __

In [25]:
```
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

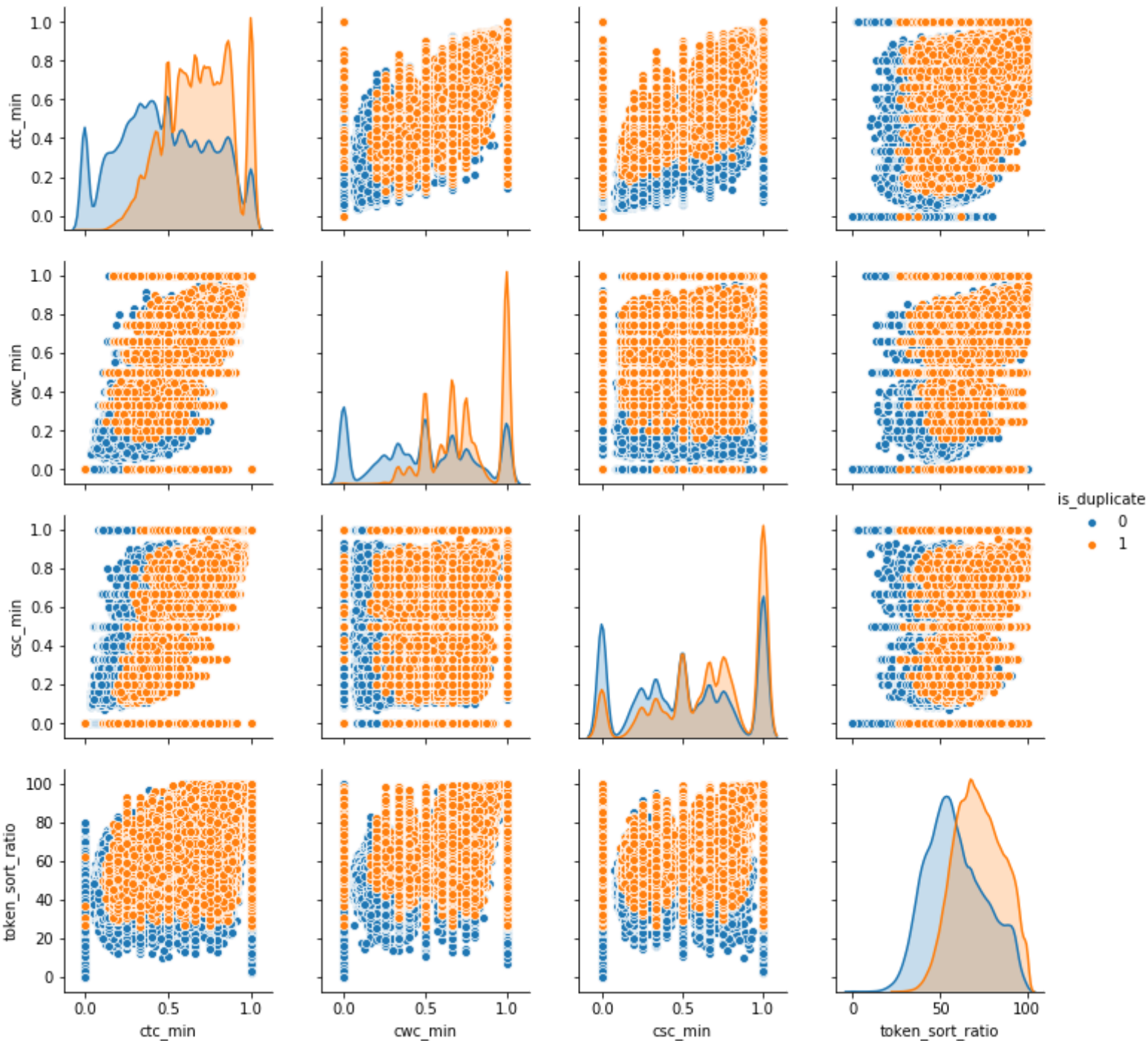
__ Word Clouds generated from non duplicate pair question's text __

In [26]:
```
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()
```

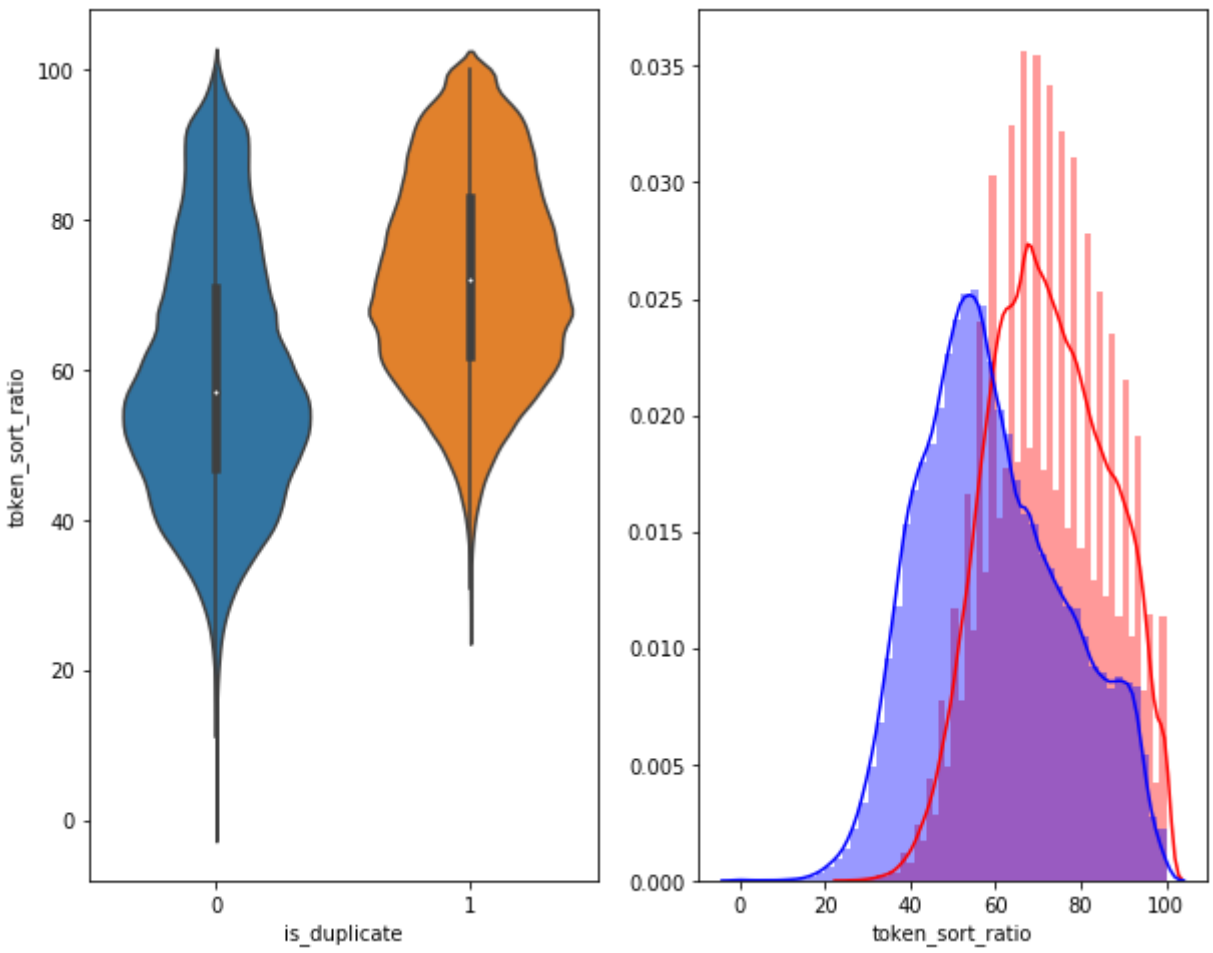
Word Cloud for non-Duplicate Question pairs:

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

In [27]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['
plt.show()
```



In [28]:
```
# 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()
```



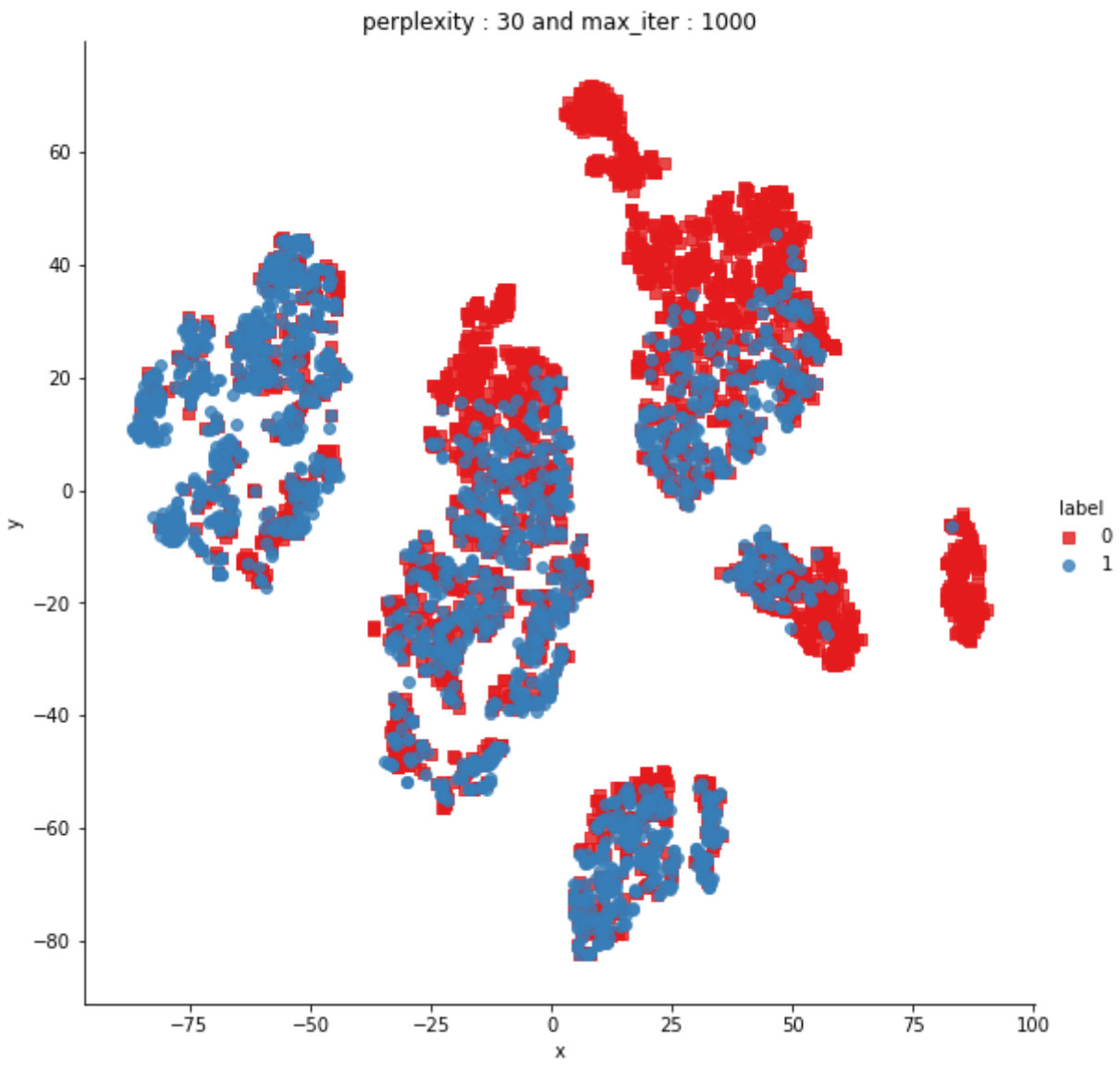
3.5.2 Visualization

In [29]:
```
# 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' , 'l
y = dfp_subsampled['is_duplicate'].values
```

In [30]:
```
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 0.037s...
[t-SNE] Computed neighbors for 5000 samples in 0.381s...
[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 0.230s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iterations in 2.688s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iterations in 2.066s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iterations in 2.063s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iterations in 2.047s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iterations in 2.054s)
[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 2.185s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iterations in 2.176s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iterations in 2.016s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iterations in 2.005s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iterations in 1.986s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iterations in 1.997s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iterations in 2.008s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iterations in 2.026s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iterations in 2.022s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iterations in 2.040s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iterations in 2.018s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iterations in 1.997s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iterations in 2.017s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iterations in 2.033s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iterations in 2.025s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

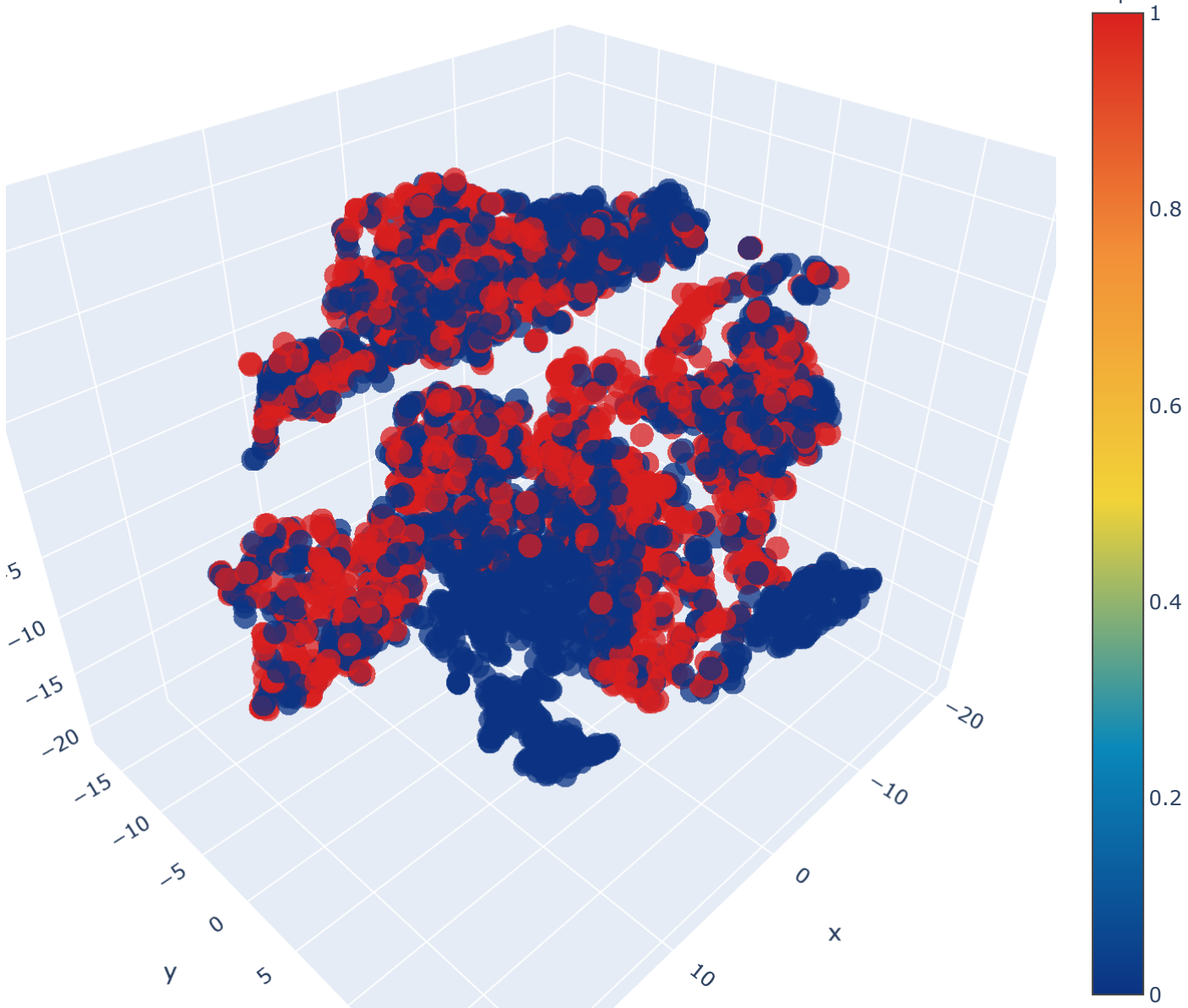
In [31]:
```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# 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 [32]:
```
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 0.009s...
[t-SNE] Computed neighbors for 5000 samples in 0.354s...
[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 0.191s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 iterations in 9.055s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (50 iterations in 5.008s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (50 iterations in 4.492s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (50 iterations in 4.507s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (50 iterations in 4.420s)
[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 5.600s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50 iterations in 7.268s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50 iterations in 7.405s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50 iterations in 7.393s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50 iterations in 7.622s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50 iterations in 7.093s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50 iterations in 7.123s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50 iterations in 7.140s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50 iterations in 7.178s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50 iterations in 7.233s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50 iterations in 7.510s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50 iterations in 7.367s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50 iterations in 7.259s)
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50 iterations in 7.154s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (50 iterations in 7.321s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [33]:
```
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')
```



3.6 Featurizing text data with tfidf weighted word-vectors

In [34]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
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
import pandas as pd
import numpy as np
from tqdm import tqdm

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

In [35]:
```
# 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 [36]:
```
df.head()
```

Out[36]:

	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 [37]:
```
#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 [38]:
```
dfnlp.head(2)
```

Out[38]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	first_word_eq	abs_len_d
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	1.0	2
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	1.0	5

2 rows × 21 columns

In [39]:
```
df1  = dfnlp.merge(dfppro, on='id',how='left')
```

In [40]:
```
df1 = df1.drop(['qid1_x','qid2_x'],axis=1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [41]:
```
df1.shape
```

Out[41]:
```
(404290, 35)
```

In [42]:
```
df4  = df1.merge(df3, on='id',how='left')
```

In [43]:
```
df4 = df4.sample(n=50000)
```

In [44]:
```
df4.shape
```

Out[44]:
```
(50000, 35)
```

In [45]:
```
y_true = df4['is_duplicate_x']
```

In [46]:
```
y_true
```

Out[46]:
```
224177    1
126111    0
122602    1
302165    1
148546    1
         ..
20430     0
375555    0
337622    0
339551    0
110283    1
Name: is_duplicate_x, Length: 50000, dtype: int64
```

In [47]:
```
df4 = df4.drop(['is_duplicate_x'],axis=1)
```

In [48]:
```
df4.shape
```

Out[48]:
```
(50000, 34)
```

In [49]:
```
from sklearn.model_selection import train_test_split
```

In [50]:
```
X_train,X_test, Y_train, Y_test = train_test_split(df4, y_true, stratify=y_true, test_size=0.3)
```

In [51]:
```
X_train.shape,X_test.shape
```

Out[51]:
```
((35000, 34), (15000, 34))
```

In [52]:
```
# Filling the null values with ' '
X_train = X_train.fillna(' ')
nan_rows1 = X_train[X_train.isnull().any(1)]
print (nan_rows1)

# Filling the null values with ' '
X_test = X_test.fillna(' ')
nan_rows2 = X_test[X_test.isnull().any(1)]
print (nan_rows2)
```

```
Empty DataFrame
Columns: [id, question1_x, question2_x, 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, qi
d1_y, qid2_y, question1_y, question2_y, is_duplicate_y, freq_qid1, freq_qid2, q1len, q2len, q1_n_words, q2_n_words, wor
d_Common, word_Total, word_share, freq_q1+q2, freq_q1-q2]
Index: []

[0 rows x 34 columns]
Empty DataFrame
Columns: [id, question1_x, question2_x, 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, qi
d1_y, qid2_y, question1_y, question2_y, is_duplicate_y, freq_qid1, freq_qid2, q1len, q2len, q1_n_words, q2_n_words, wor
d_Common, word_Total, word_share, freq_q1+q2, freq_q1-q2]
Index: []

[0 rows x 34 columns]
```

TFIDFW2V Vectorization on train data:

In [53]:
```
X_train['question1_x'].isnull().values.any()
```

Out[53]:
```
False
```

In [54]:
```
X_train[X_train.isnull().any(1)]
```

Out[54]:

id	question1_x	question2_x	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	freq_qid2	q1len	q2len	q1_n_words

0 rows × 34 columns

In [55]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions_train = list(X_train['question1_x']) + list(X_train['question2_x'])

tfidf_train = TfidfVectorizer(lowercase=False, )
tfidf_train.fit_transform(questions_train)

# # dict key:word and value:tf-idf score
word2tfidf_train = dict(zip(tfidf_train.get_feature_names(), tfidf_train.idf_))
```

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

In [56]:
```
(X_train['question1_x'][:5])
```

Out[56]:
```
331936    what are the causes of halitosis and stomach t...
256725    how do you add an image to a question or a pos...
39795     can a phone be charged by connecting one end o...
386985    in revenge of the sith  why does obi wan leave...
117577          what does it take to make a cryptocurrency
Name: question1_x, dtype: object
```

In [57]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
from tqdm import tqdm
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_train['question1_x'])):
    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_train[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
X_train['q1_feats_m'] = list(vecs1)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 35000/35000 [04:29<00:00, 130.09it/s]
```

In [58]:
```
vecs2 = []
for qu2 in tqdm(list(X_train['question2_x'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf_train[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
X_train['q2_feats_m'] = list(vecs2)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 35000/35000 [04:27<00:00, 130.82it/s]
```

In [59]:
```
# data before preprocessing
X_train.head(2)
```

Out[59]:

	id	question1_x	question2_x	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	q2len	q1_n_words	q2
331936	331936	what are the causes of halitosis and stomach t...	what causes halitosis	0.999950	0.666644	0.999900	0.166664	0.999967	0.299997	1.0	...	22	10	
256725	256725	how do you add an image to a question or a pos...	how can i add photos to my question in quora	0.749981	0.599988	0.333328	0.249997	0.499995	0.357140	1.0	...	45	14	

2 rows × 36 columns

In [60]:
```
X_train_q1 = pd.DataFrame(X_train.q1_feats_m.values.tolist(), index= X_train.index)
X_train_q2 = pd.DataFrame(X_train.q2_feats_m.values.tolist(), index= X_train.index)
```

In [61]:
```
X_train.shape
```

Out[61]:
```
(35000, 36)
```

In [ ]:
```
```

In [62]:
```
X_train_q1['id']=X_train['id']
X_train_q2['id']=X_train['id']
df1  = X_train_q1.merge(X_train_q2, on='id',how='left')
X_train  = X_train.merge(df1, on='id',how='left')
```

In [63]:
```
X_train.shape
```

Out[63]:
```
(35000, 228)
```

TFIDFW2V Vectorization on test data

In [64]:
```
X_test['question1_x'].isnull().values.any()
```

Out[64]:
```
False
```

In [65]:
```
X_test[X_test.isnull().any(1)]
```

Out[65]:

id	question1_x	question2_x	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	freq_qid2	q1len	q2len	q1_n_words

0 rows × 34 columns

In [66]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions_test = list(X_test['question1_x']) + list(X_test['question2_x'])

# tfidf_train = TfidfVectorizer(lowercase=False, )
tfidf_train.transform(questions_test)

# # dict key:word and value:tf-idf score
word2tfidf_test = dict(zip(tfidf_train.get_feature_names(), tfidf_train.idf_))
```

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

In [67]:
```
(X_test['question1_x'][12980:12995])
```

Out[67]:
```
188308    how can i get a complete list of all my gmail ...
30801                      which are royal songs in kannada
385391         how good is rutgers mbs   analytics program
152752        how do i stop my dog from chewing on its fur
314797                  how can i catch my husband cheating
97845                         why does india oppose cpec
275700                   how can you increase your height
238408            how do i stop talking to myself out loud
293803      what is the best vr headset available in india
250747    which indian colleges offer astronomy and astr...
256873           what are the best online coding bootcamps
238673    do straight women like gay porn  why  or why not
211556    what are the names of the best three universit...
148962            what is the easiest way to learn chinese
392854    do surgeons really multitask like they do on  ...
Name: question1_x, dtype: object
```

In [68]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_test['question1_x'])):
    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_test[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
X_test['q1_feats_m'] = list(vecs1)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 15000/15000 [01:57<00:00, 127.28it/s]
```

In [69]:
```
X_test.shape
```

Out[69]:
```
(15000, 35)
```

In [70]:
```
vecs2 = []
for qu2 in tqdm(list(X_test['question2_x'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf_test[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
X_test['q2_feats_m'] = list(vecs2)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 15000/15000 [01:55<00:00, 129.79it/s]
```

In [71]:
```
X_test_q1 = pd.DataFrame(X_test.q1_feats_m.values.tolist(), index= X_test.index)
X_test_q2 = pd.DataFrame(X_test.q2_feats_m.values.tolist(), index= X_test.index)
```

In [72]:
```
X_test.shape
```

Out[72]:
```
(15000, 36)
```

In [73]:
```

X_test_q1['id']=X_test['id']
X_test_q2['id']=X_test['id']
df2  = X_test_q1.merge(X_test_q2, on='id',how='left')
X_test  = X_test.merge(df2, on='id',how='left')
```

In [74]:
```
X_train.shape,X_test.shape
```

Out[74]:
```
((35000, 228), (15000, 228))
```

Storing final features and their targets with respective splitting¶

In [75]:
```
X_train.to_pickle("X_train.txt")
Y_train.to_pickle("y_train.txt")
X_test.to_pickle("X_test.txt")
Y_test.to_pickle("y_test.txt")
```

In [ ]:
```

```

In [76]:
```
X_train = X_train.drop(['id','question1_x','question2_x','q1_feats_m','q2_feats_m','qid1_y','qid2_y','question1_y','ques
```

In [77]:
```
X_test = X_test.drop(['id','question1_x','question2_x','q1_feats_m','q2_feats_m','qid1_y','qid2_y','question1_y','questi
```

In [78]:
```
X_train.columns
```

Out[78]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       ...
       '86_y', '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
       '95_y'],
      dtype='object', length=218)
```

In [79]:
```
X_test.columns
```

Out[79]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       ...
       '86_y', '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
       '95_y'],
      dtype='object', length=218)
```

In [80]:
```
cols = list(X_train.columns)
print(cols[:10])
```

```
['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq', 'abs_len_diff', 'me
an_len']
```

In [81]:
```
cols = list(X_test.columns)
print(cols[:10])
```

```
['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq', 'abs_len_diff', 'me
an_len']
```

4. Machine Learning Models

In [82]:
```
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 [83]:
```
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:  0.6299142857142858 Class 1:  0.3700857142857143
---------- Distribution of output variable in train data ----------
Class 0:  0.37006666666666665 Class 1:  0.37006666666666665
```

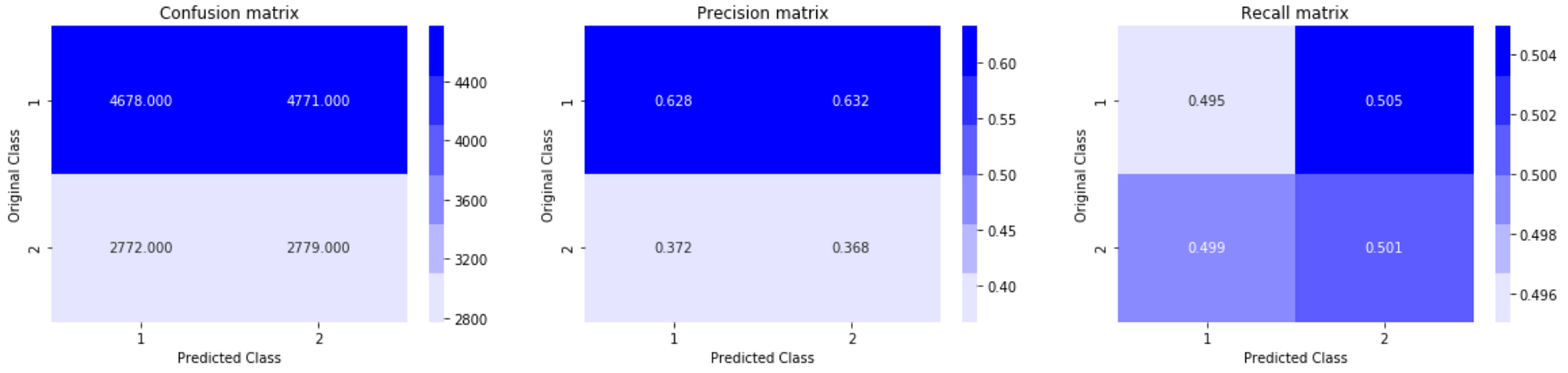
In [84]:

```
# 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()
```

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

In [85]:
```
# 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 0.8984356418372561
```



Logistic Regression with hyperparameter tuning

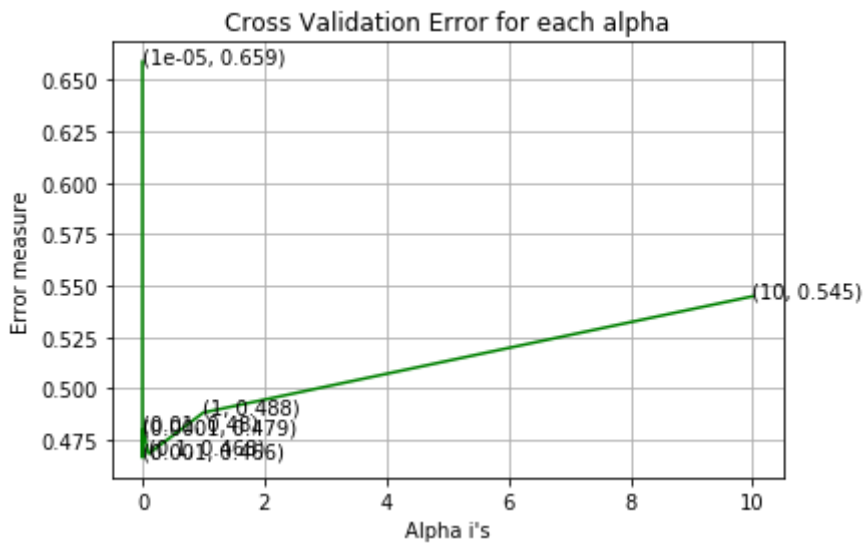
In [86]:
```
X_train.shape,Y_train.shape
```

Out[86]:
```
((35000, 218), (35000,))
```

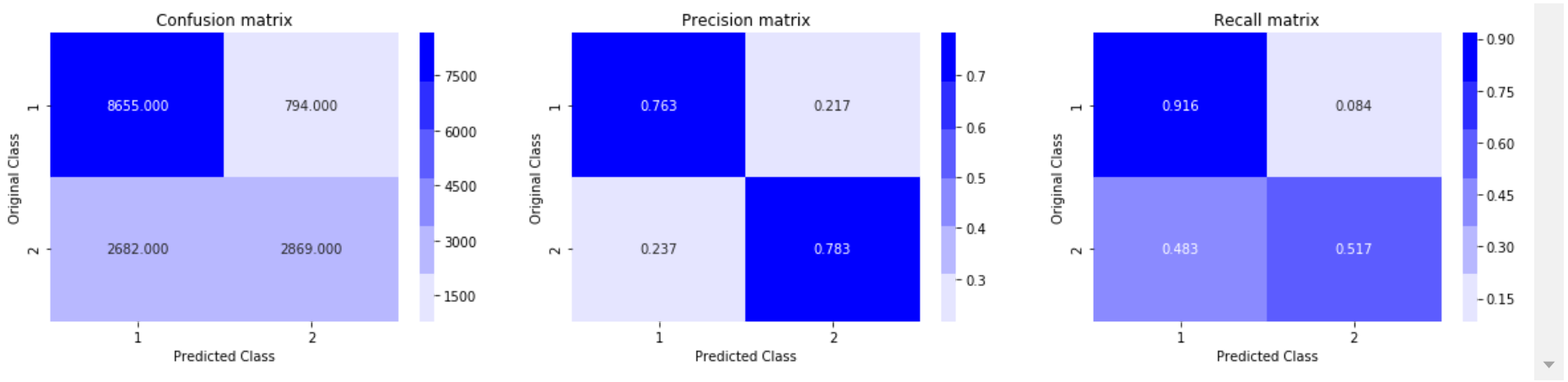
In [87]:

```
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
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, 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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train, 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
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.c
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: 0.658991155931641
For values of alpha =  0.0001 The log loss is: 0.4787245539861891
For values of alpha =  0.001 The log loss is: 0.46647389598404443
For values of alpha =  0.01 The log loss is: 0.48016258885422103
For values of alpha =  0.1 The log loss is: 0.46835018201682194
For values of alpha =  1 The log loss is: 0.48838430673165445
For values of alpha =  10 The log loss is: 0.5448923546663916
```



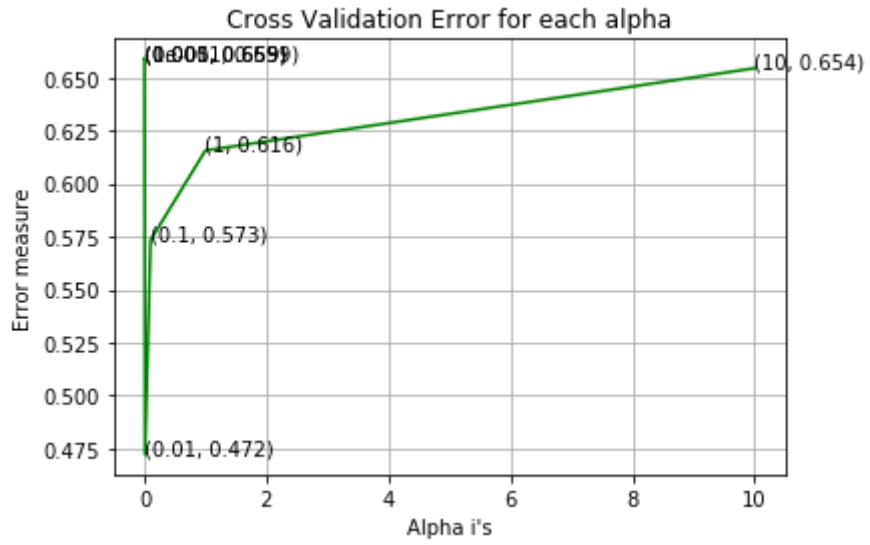

```
For values of best alpha =  0.001 The train log loss is: 0.46973721515094374
For values of best alpha =  0.001 The test log loss is: 0.46647389598404443
Total number of data points : 15000
```



Linear SVM with hyperparameter tuning

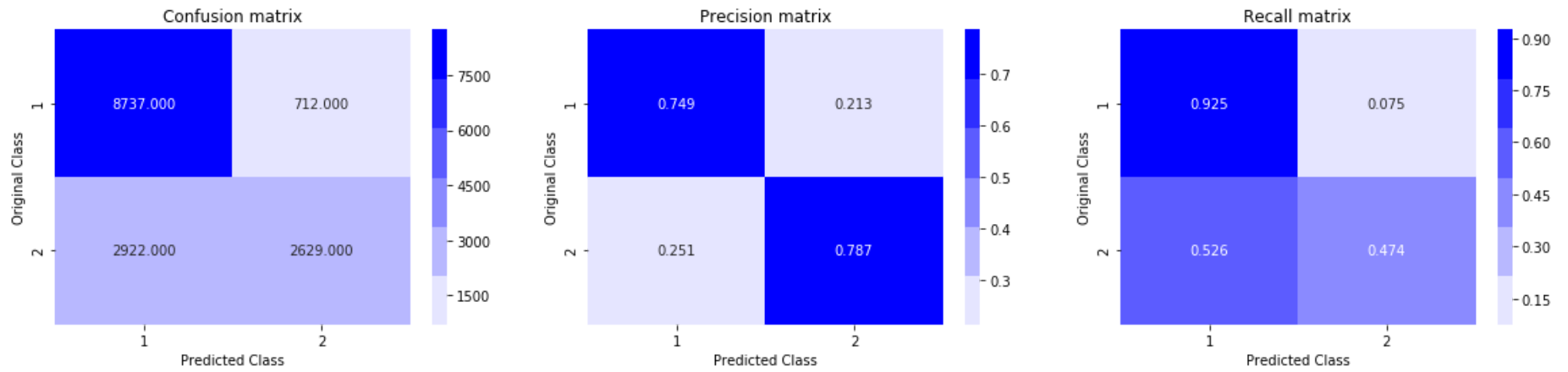
In [88]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, 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.classes_, eps=1e-15))

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

```
For values of alpha =  1e-05 The log loss is: 0.658991155931641
For values of alpha =  0.0001 The log loss is: 0.658991155931641
For values of alpha =  0.001 The log loss is: 0.658991155931641
For values of alpha =  0.01 The log loss is: 0.472105428357441
For values of alpha =  0.1 The log loss is: 0.5732625766488726
For values of alpha =  1 The log loss is: 0.615634701111975
For values of alpha =  10 The log loss is: 0.65443996068772
```



In [89]:
```
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
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.c
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 best alpha =  0.01 The train log loss is: 0.4821215147301954
For values of best alpha =  0.01 The test log loss is: 0.472105428357441
Total number of data points : 15000
```



XGBoost

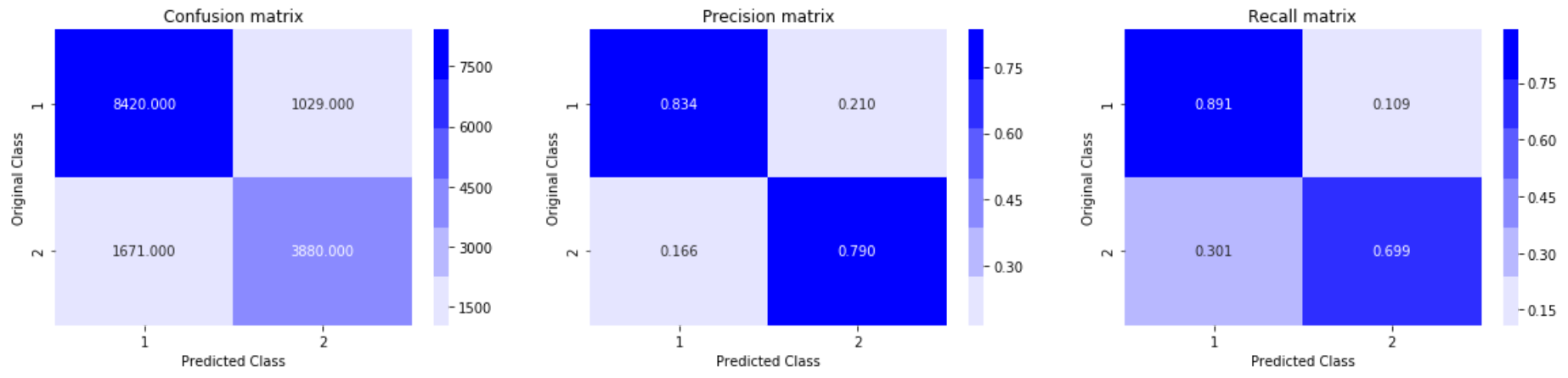
In [142]:
```
#https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/
```

In [94]:
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

d_train = xgb.DMatrix(X_train, label=Y_train)
d_test = xgb.DMatrix(X_test, label=Y_test)

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

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

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

```
[0]     train-logloss:0.684909  valid-logloss:0.684877
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:0.616633  valid-logloss:0.61609
[20]    train-logloss:0.566071  valid-logloss:0.565248
[30]    train-logloss:0.527791  valid-logloss:0.526729
[40]    train-logloss:0.498328  valid-logloss:0.497193
[50]    train-logloss:0.474918  valid-logloss:0.473771
[60]    train-logloss:0.456505  valid-logloss:0.455425
[70]    train-logloss:0.441576  valid-logloss:0.440657
[80]    train-logloss:0.429644  valid-logloss:0.42882
[90]    train-logloss:0.419932  valid-logloss:0.419218
[100]   train-logloss:0.41162   valid-logloss:0.411126
[110]   train-logloss:0.404771  valid-logloss:0.40457
[120]   train-logloss:0.399394  valid-logloss:0.399334
[130]   train-logloss:0.39471   valid-logloss:0.394863
[140]   train-logloss:0.390766  valid-logloss:0.391292
[150]   train-logloss:0.387278  valid-logloss:0.388236
[160]   train-logloss:0.383959  valid-logloss:0.385173
[170]   train-logloss:0.381179  valid-logloss:0.382726
[180]   train-logloss:0.378631  valid-logloss:0.380523
[190]   train-logloss:0.376369  valid-logloss:0.378615
[200]   train-logloss:0.374408  valid-logloss:0.376999
[210]   train-logloss:0.372399  valid-logloss:0.37539
[220]   train-logloss:0.37067   valid-logloss:0.373933
[230]   train-logloss:0.368851  valid-logloss:0.372513
[240]   train-logloss:0.367053  valid-logloss:0.37113
[250]   train-logloss:0.36523   valid-logloss:0.369719
[260]   train-logloss:0.363488  valid-logloss:0.368424
[270]   train-logloss:0.361892  valid-logloss:0.367307
[280]   train-logloss:0.360327  valid-logloss:0.366223
[290]   train-logloss:0.358832  valid-logloss:0.365202
[300]   train-logloss:0.357406  valid-logloss:0.364219
[310]   train-logloss:0.356063  valid-logloss:0.363448
[320]   train-logloss:0.354797  valid-logloss:0.362765
[330]   train-logloss:0.353547  valid-logloss:0.362124
[340]   train-logloss:0.352363  valid-logloss:0.361536
[350]   train-logloss:0.351058  valid-logloss:0.360853
[360]   train-logloss:0.349871  valid-logloss:0.360273
[370]   train-logloss:0.348648  valid-logloss:0.359694
[380]   train-logloss:0.347594  valid-logloss:0.359182
[390]   train-logloss:0.346466  valid-logloss:0.358679
[399]   train-logloss:0.345566  valid-logloss:0.358288
The test log loss is: 0.35828752034377154
```

In [95]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(Y_test, predicted_y)
```

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



Assignments

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

In [104]:
```
df1  = dfnlp.merge(dfppro, on='id',how='left')


df1 = df1.drop(['qid1_x','qid2_x'],axis=1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [105]:
```
df4  = df1.merge(df3, on='id',how='left')
```

In [106]:
```
df4.shape
```

Out[106]:
```
(404290, 35)
```

In [107]:
```
df4 = df4.sample(n=50000)
```

In [108]:
```
y_true = df4['is_duplicate_x']
```

In [109]:
```
y_true[:3]
```

Out[109]:
```
139253    0
381731    0
7678      0
Name: is_duplicate_x, dtype: int64
```

In [110]:
```
df4 = df4.drop(['is_duplicate_x'],axis=1)
```

In [111]:
```
df4.shape
```

Out[111]:
```
(50000, 34)
```

In [112]:
```
X_train,X_test, y_train, y_test = train_test_split(df4, y_true, stratify=y_true, test_size=0.3)
```

In [114]:
```
# Filling the null values with ' '
X_train = X_train.fillna(' ')
nan_rows1 = X_train[X_train.isnull().any(1)]

# Filling the null values with ' '
X_test = X_test.fillna(' ')
nan_rows2 = X_test[X_test.isnull().any(1)]
```

Preparing data

Tfidf vectorization

In [115]:
```
# Train Data
from sklearn.feature_extraction.text import TfidfVectorizer
questions_train = list(X_train['question1_x']) + list(X_train['question2_x'])
vectorizer_tfidf_ques = TfidfVectorizer(lowercase=False,min_df=10)
vectorizer_tfidf_ques.fit(questions_train)

q1_tfidf_train = vectorizer_tfidf_ques.transform(X_train['question1_x'])
q2_tfidf_train = vectorizer_tfidf_ques.transform(X_train['question2_x'])
print("Shape of matrix after q1_tfidf_train ",q1_tfidf_train.shape)
print("Shape of matrix after q2_tfidf_train ",q2_tfidf_train.shape)
```

```
Shape of matrix after q1_tfidf_train  (35000, 5225)
Shape of matrix after q2_tfidf_train  (35000, 5225)
```

In [116]:
```
X_train = X_train.drop(['id','question1_x','question2_x','qid1_y','qid2_y','question1_y','question2_y','is_duplicate_y']
```

In [117]:
```
X_train = hstack((X_train,q1_tfidf_train,q2_tfidf_train))
```

In [118]:
```
X_train.shape
```

Out[118]:
```
(35000, 10476)
```

In [119]:
```
# Test data
from sklearn.feature_extraction.text import TfidfVectorizer

q1_tfidf_test = vectorizer_tfidf_ques.transform(X_test['question1_x'])
q2_tfidf_test = vectorizer_tfidf_ques.transform(X_test['question2_x'])
print("Shape of matrix after q1_tfidf_test ",q1_tfidf_test.shape)
print("Shape of matrix after q2_tfidf_test ",q2_tfidf_test.shape)
```

```
Shape of matrix after q1_tfidf_test  (15000, 5225)
Shape of matrix after q2_tfidf_test  (15000, 5225)
```

In [120]:
```
X_test = X_test.drop(['id','question1_x','question2_x','qid1_y','qid2_y','question1_y','question2_y','is_duplicate_y'],a
```

In [121]:
```
X_test = hstack((X_test,q1_tfidf_test,q2_tfidf_test))
```

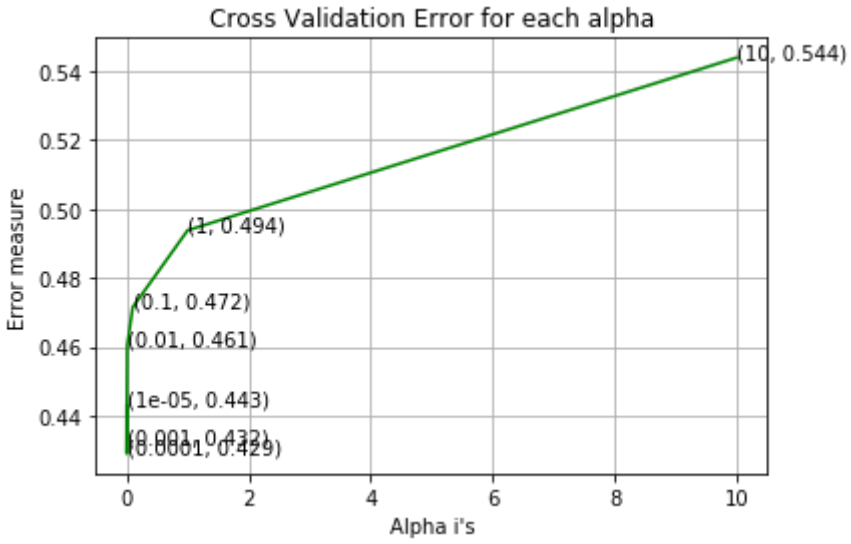
In [122]:
```
X_test.shape
```

Out[122]:
```
(15000, 10476)
```

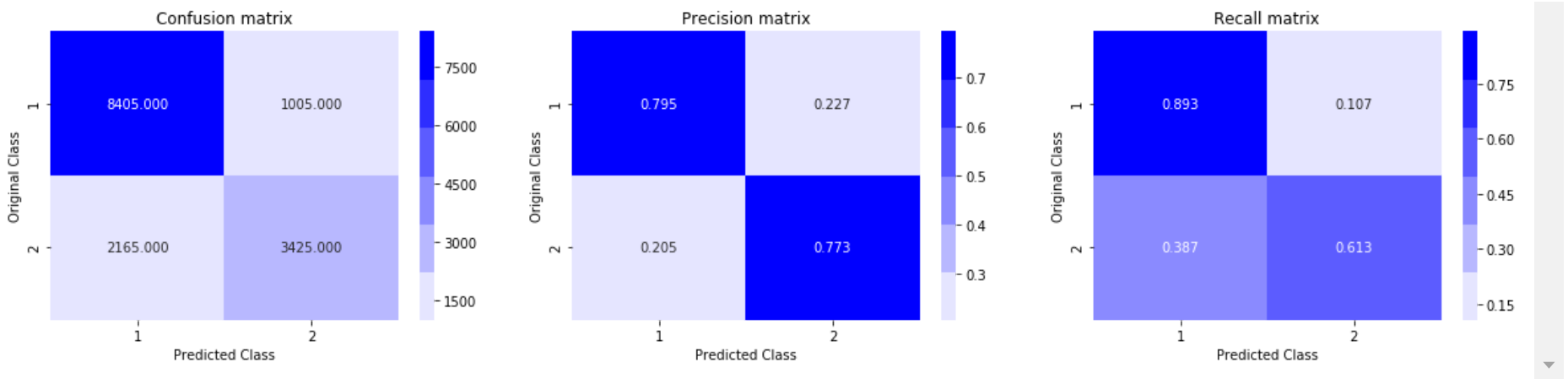
Logistic Regression with hyperparameter tuning

In [124]:
```
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
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, 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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train, 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
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.c
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: 0.44288826648849694
For values of alpha =  0.0001 The log loss is: 0.42903570730872514
For values of alpha =  0.001 The log loss is: 0.4321813986563108
For values of alpha =  0.01 The log loss is: 0.46055105378256
For values of alpha =  0.1 The log loss is: 0.47150543733571865
For values of alpha =  1 The log loss is: 0.4938091058632754
For values of alpha =  10 The log loss is: 0.5438953611570855
```



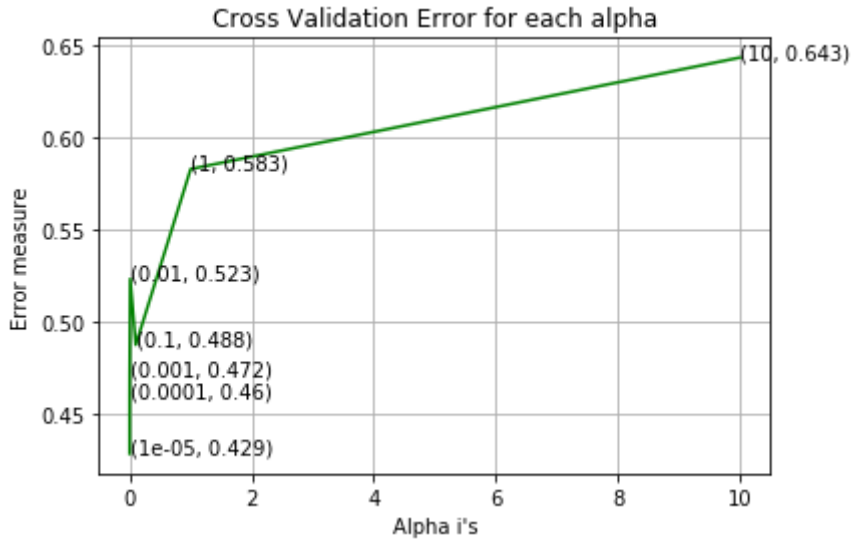

```
For values of best alpha =  0.0001 The train log loss is: 0.42013286955378254
For values of best alpha =  0.0001 The test log loss is: 0.42903570730872514
Total number of data points : 15000
```



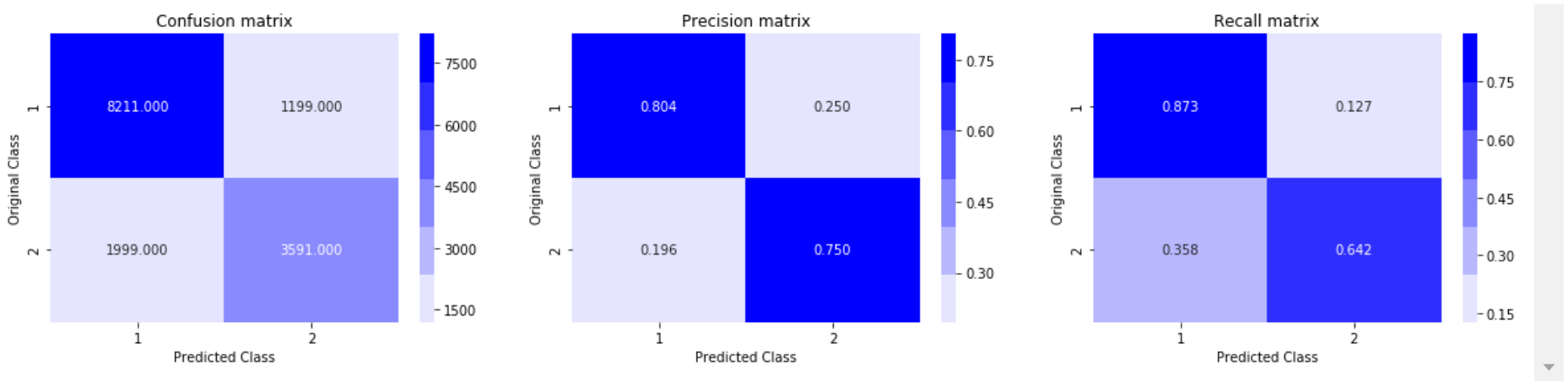
Linear SVM with hyperparameter tuning

In [125]:
```
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
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, 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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(X_train, 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
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.c
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: 0.4289075608186739
For values of alpha =  0.0001 The log loss is: 0.459524103677009
For values of alpha =  0.001 The log loss is: 0.471917706443534
For values of alpha =  0.01 The log loss is: 0.5234298246456801
For values of alpha =  0.1 The log loss is: 0.48775348979197436
For values of alpha =  1 The log loss is: 0.5828591737934609
For values of alpha =  10 The log loss is: 0.643146867575327
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4174833091437958
For values of best alpha =  1e-05 The test log loss is: 0.4289075608186739
Total number of data points : 15000
```



XGBoost

Hyperperparameter tuniny using randomsearchcv

In [130]:
```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
params = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'eta' : [0.01, 0.02, 0.05, 0.1]
}

xgb = xgb.XGBClassifier()
random_search = RandomizedSearchCV(xgb, param_distributions=params, scoring='neg_log_loss', n_jobs=-1, verbose=10, rando
random_search.fit(X_train, y_train)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   43.7s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  2.7min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  3.1min remaining:   57.3s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  3.6min remaining:   23.9s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  4.0min finished
```

Out[130]:
```
RandomizedSearchCV(cv='warn', 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='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'eta': [0.01, 0.02, 0.05, 0.1],
                                        'max_depth': [3, 4, 5, 6, 7, 8]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=True, scoring='neg_log_loss', verbose=10)
```

In [133]:
```
best_params=random_search.best_params_
print(best_params)
```

```
{'max_depth': 8, 'eta': 0.02}
```

In [134]:
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 8

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, 500, 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))
```

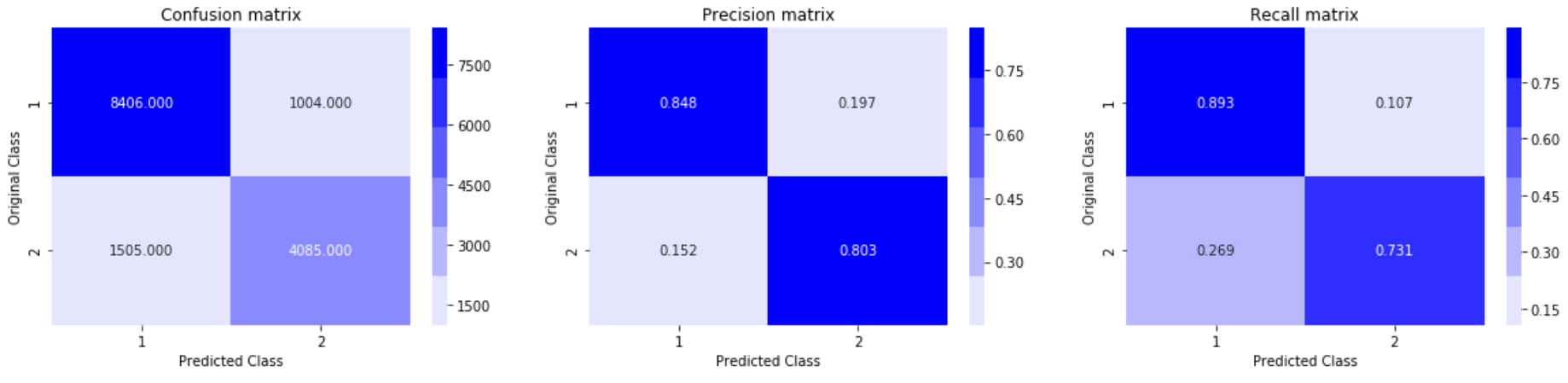
```
[0]     train-logloss:0.682824  valid-logloss:0.683316
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.597841  valid-logloss:0.602615
[20]    train-logloss:0.536572  valid-logloss:0.54521
[30]    train-logloss:0.490637  valid-logloss:0.502877
[40]    train-logloss:0.455194  valid-logloss:0.470613
[50]    train-logloss:0.427337  valid-logloss:0.446014
[60]    train-logloss:0.405019  valid-logloss:0.426763
[70]    train-logloss:0.386646  valid-logloss:0.411366
[80]    train-logloss:0.371332  valid-logloss:0.398992
[90]    train-logloss:0.358436  valid-logloss:0.388667
[100]   train-logloss:0.348487  valid-logloss:0.380709
[110]   train-logloss:0.340467  valid-logloss:0.374282
[120]   train-logloss:0.33379   valid-logloss:0.369091
[130]   train-logloss:0.327475  valid-logloss:0.364613
[140]   train-logloss:0.322556  valid-logloss:0.361117
[150]   train-logloss:0.318438  valid-logloss:0.358193
[160]   train-logloss:0.314513  valid-logloss:0.355588
[170]   train-logloss:0.311547  valid-logloss:0.35353
[180]   train-logloss:0.308057  valid-logloss:0.351519
[190]   train-logloss:0.305396  valid-logloss:0.34993
[200]   train-logloss:0.30328   valid-logloss:0.348699
[210]   train-logloss:0.301205  valid-logloss:0.347589
[220]   train-logloss:0.299314  valid-logloss:0.346584
[230]   train-logloss:0.297621  valid-logloss:0.345856
[240]   train-logloss:0.29609   valid-logloss:0.345177
[250]   train-logloss:0.294771  valid-logloss:0.344512
[260]   train-logloss:0.293627  valid-logloss:0.344025
[270]   train-logloss:0.292462  valid-logloss:0.343478
[280]   train-logloss:0.291172  valid-logloss:0.342918
[290]   train-logloss:0.289813  valid-logloss:0.342347
[300]   train-logloss:0.288705  valid-logloss:0.341938
[310]   train-logloss:0.287312  valid-logloss:0.341462
[320]   train-logloss:0.286436  valid-logloss:0.341146
[330]   train-logloss:0.285346  valid-logloss:0.340762
[340]   train-logloss:0.284454  valid-logloss:0.34051
[350]   train-logloss:0.283658  valid-logloss:0.340265
[360]   train-logloss:0.282915  valid-logloss:0.339997
[370]   train-logloss:0.282014  valid-logloss:0.339723
[380]   train-logloss:0.281168  valid-logloss:0.339503
[390]   train-logloss:0.2804    valid-logloss:0.339268
[400]   train-logloss:0.279509  valid-logloss:0.339048
[410]   train-logloss:0.278806  valid-logloss:0.338874
[420]   train-logloss:0.278007  valid-logloss:0.338691
[430]   train-logloss:0.277284  valid-logloss:0.338489
[440]   train-logloss:0.276625  valid-logloss:0.338285
[450]   train-logloss:0.27587   valid-logloss:0.338013
[460]   train-logloss:0.275292  valid-logloss:0.337865
[470]   train-logloss:0.274675  valid-logloss:0.337707
[480]   train-logloss:0.27405   valid-logloss:0.337525
[490]   train-logloss:0.273365  valid-logloss:0.337392
[499]   train-logloss:0.272802  valid-logloss:0.337233
The test log loss is: 0.33723252494984773
```

In [135]:
```
print("The test log loss is:",log_loss(y_test, predict_y, labels=random_search.classes_, eps=1e-15))
```

```
The test log loss is: 0.33723252494984773
```

In [136]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

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



summary

In [141]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model","Test log loss"]
x.add_row(["TFIDF", "Logistic Regression",0.42])
x.add_row(["TFIDF", "Linear SVM",0.42])
x.add_row(["TFIDF", "XGBoost" ,0.33])
print(x)
```

```
+------------+---------------------+---------------+
| Vectorizer |        Model        | Test log loss |
+------------+---------------------+---------------+
|   TFIDF    | Logistic Regression |      0.42     |
|   TFIDF    |      Linear SVM     |      0.42     |
|   TFIDF    |       XGBoost       |      0.33     |
+------------+---------------------+---------------+
```

XgBoost works best for this dataset

In [ ]: