Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

Problem Statement

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

1.2 Sources/Useful Links

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

Useful Links

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

In [ ]:

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
from tqdm import tqdm

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
import spacy
```

In [2]:
```
data=pd.read_csv('train.csv')
```

In [3]:
```
data.shape
```

Out[3]: (404290, 6)

In [4]:
```
data.head(2)
```

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

In [5]:
```
data.loc[data['is_duplicate']==1]
```

Out[5]:

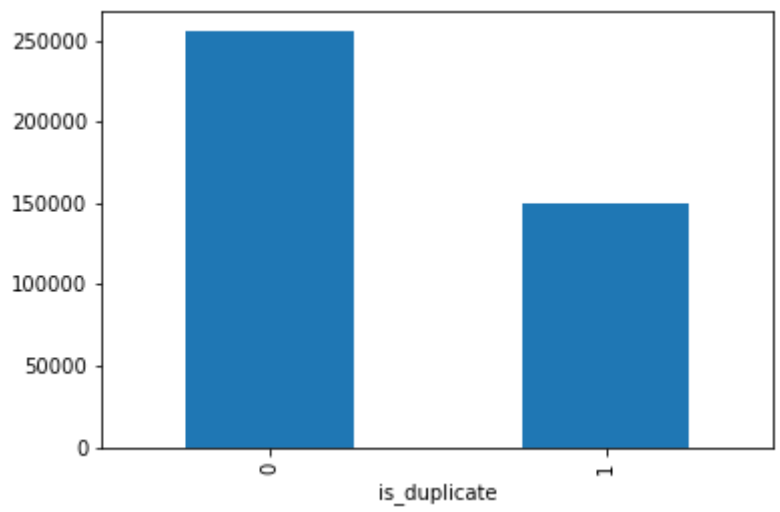
	id	qid1	qid2	question1	question2	is_duplicate
5	5	11	12	Astrology: I am a Capricorn Sun Cap moon and c...	I'm a triple Capricorn (Sun, Moon and ascendan...	1
7	7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1
11	11	23	24	How do I read and find my YouTube comments?	How can I see all my Youtube comments?	1
12	12	25	26	What can make Physics easy to learn?	How can you make physics easy to learn?	1
13	13	27	28	What was your first sexual experience like?	What was your first sexual experience?	1
...	...	...	...	...	...	...
404280	404280	537922	537923	What are some outfit ideas to wear to a frat p...	What are some outfit ideas wear to a frat them...	1
404281	404281	99131	81495	Why is Manaphy childish in Pokémon Ranger and ...	Why is Manaphy annoying in Pokemon ranger and ...	1
404282	404282	1931	16773	How does a long distance relationship work?	How are long distance relationships maintained?	1
404284	404284	537926	537927	What does Jainism say about homosexuality?	What does Jainism say about Gays and Homosexua...	1
404286	404286	18840	155606	Do you believe there is life after death?	Is it true that there is life after death?	1

149263 rows × 6 columns

In [6]:
```
f=list(data['is_duplicate'].value_counts())
```

In [7]:
```
data.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[7]:
```
<AxesSubplot:xlabel='is_duplicate'>
```



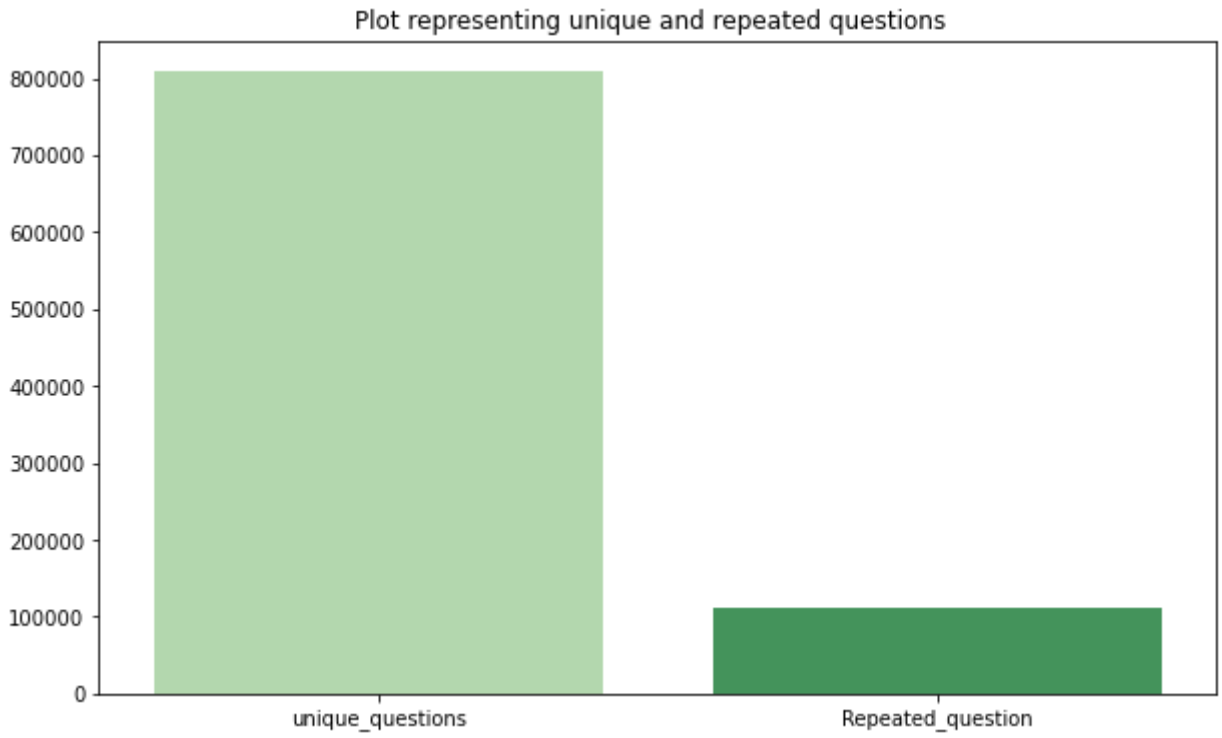
In [8]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format((f[0]/sum(f)*100)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format((f[1]/sum(f)*100)))
```

```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08021469737069%

~> Question pairs are Similar (is_duplicate = 1):
   36.9197853026293%
```

In [9]:
```
uniqid=pd.Series(data['qid1'].to_list()+data['qid2'].to_list())
print ('Total number of  Unique Questions are:')
print (len(np.unique(uniqid)))
more_once=np.sum(uniqid.value_counts()>1)

print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(more_once,more_once/len(np.unique(uniqid))*1
print("Maximum number items question repated: {}".format(max(uniqid.value_counts())))
```

```
Total number of  Unique Questions are:
537933
Number of unique questions that appear more than one time: 111780 (20.77953945937505%)

Maximum number items question repated: 157
```

In [10]:
```
x=["unique_questions","Repeated_question"]
y=[len(uniqid),more_once]
plt.figure(figsize=(10, 6))
plt.title ("Plot representing unique and repeated questions  ")
sns.barplot(x,y, palette = "Greens")
plt.show()
```

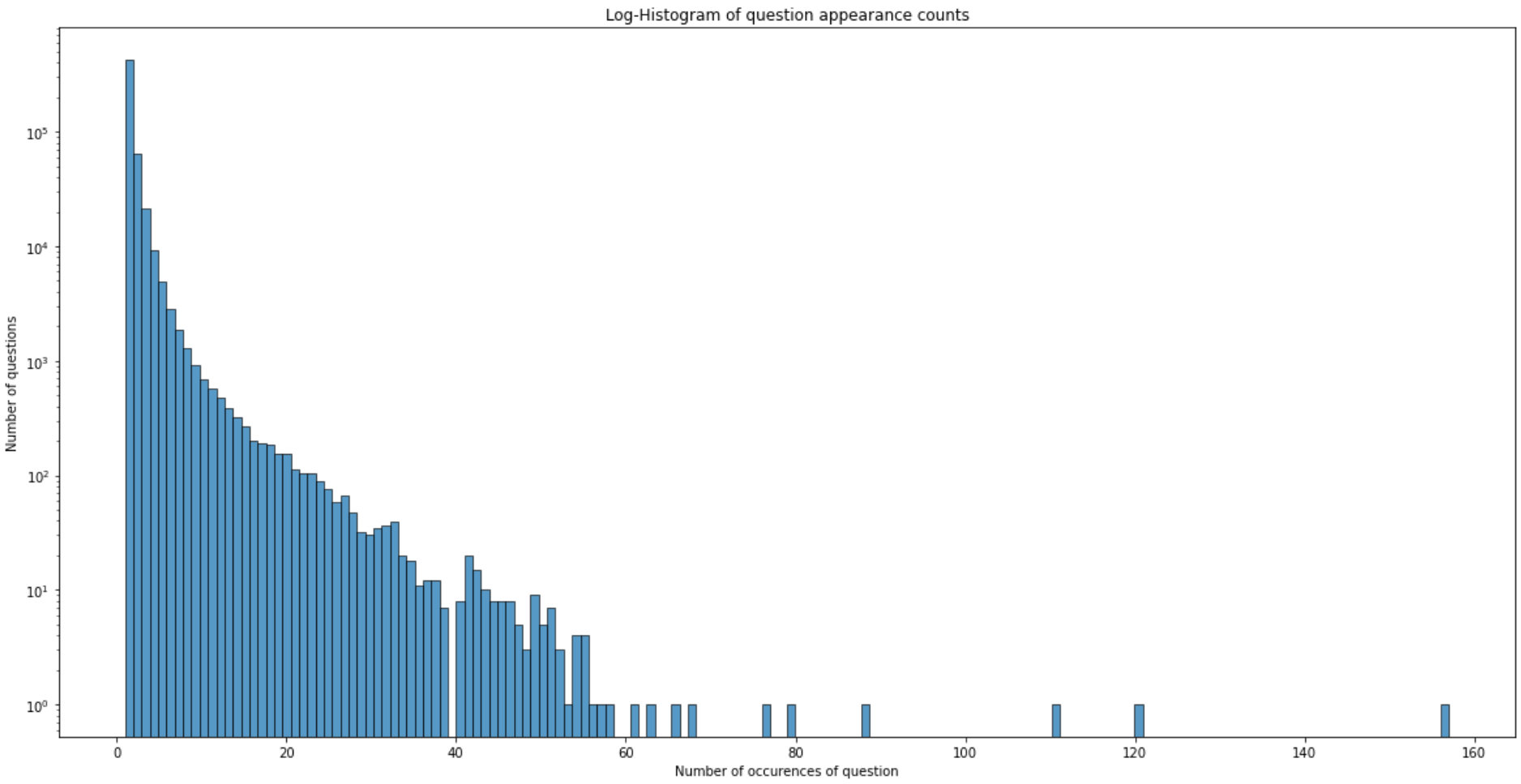


In [11]:
```
duplicate = data[data.duplicated()]
print("Duplicate Rows :{}".format(len(duplicate)))
```

```
Duplicate Rows :0
```

In [12]:
```
plt.figure(figsize=(20, 10))

sns.histplot(uniqid.value_counts(), bins=160,kde=False, palette = "Greens")

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(uniqid.value_counts())))
```

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



In [13]:
```
data.isnull().sum()
```

Out[13]:
```
id             0
qid1           0
qid2           0
question1      1
question2      2
is_duplicate   0
dtype: int64
```

In [14]:
```
data = data.fillna('')
data.isnull().sum()
```

Out[14]:
```
id             0
qid1           0
qid2           0
question1      0
question2      0
```

```
is_duplicate    0
dtype: int64
```

In [ ]:

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]:
```
data['freq_qid1'] = data.groupby('qid1')['qid1'].transform('count')
data['freq_qid2'] = data.groupby('qid2')['qid2'].transform('count')
data['q1len'] = data['question1'].str.len()
data['q2len'] = data['question2'].str.len()
data['q1_n_words'] = data['question1'].apply(lambda row: len(row.split(" ")))
data['q2_n_words'] = data['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)
data['word_Common'] = data.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))
data['word_Total'] = data.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))
data['word_share'] = data.apply(normalized_word_share, axis=1)

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

data.to_csv("data_fe_without_preprocessing_train.csv", index=False)

data.head()
```

Out[15]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14	12	10.0	23.0	0.4
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	0.2
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	0.1
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	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	20.0	0.1

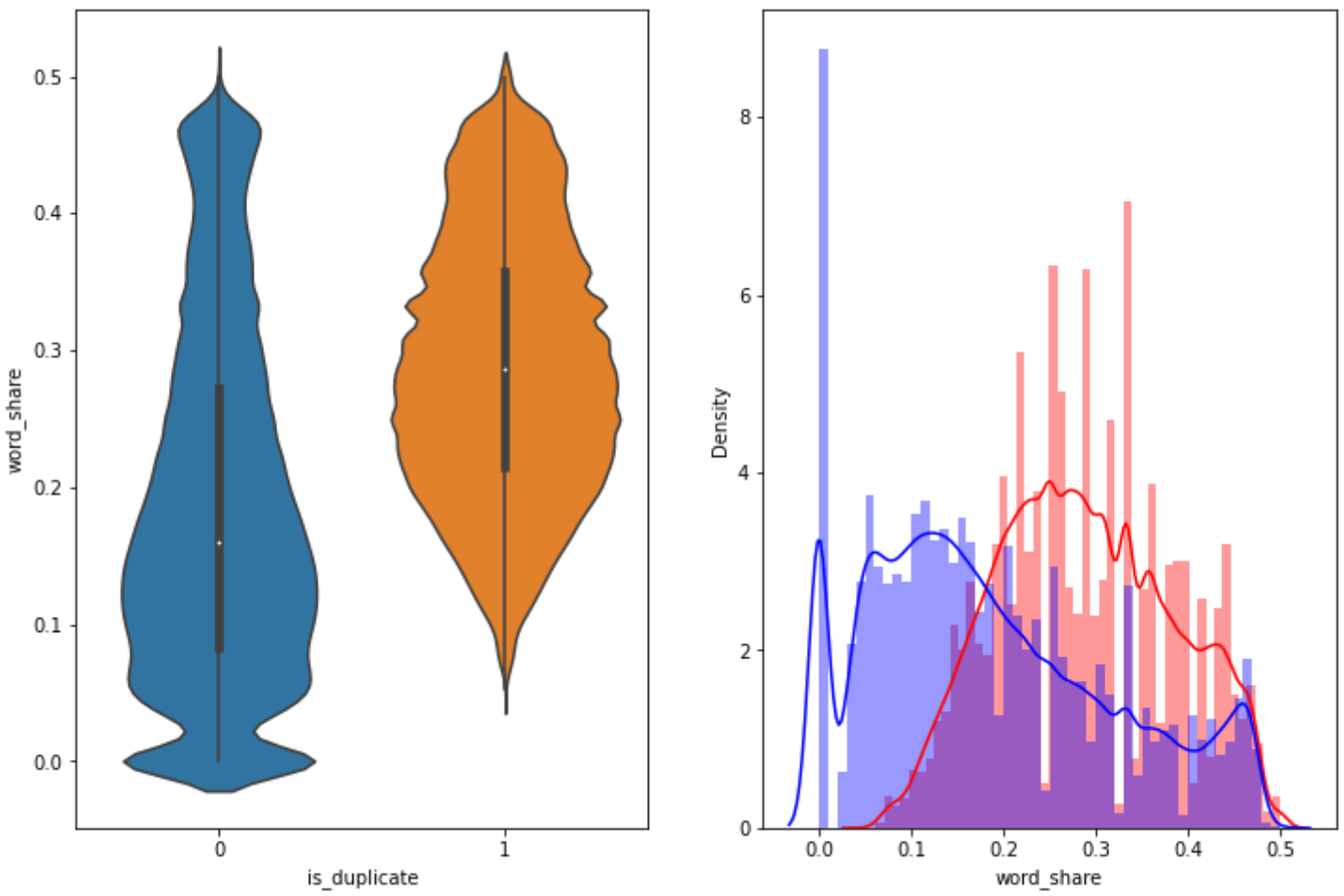
In [16]:
```
print ("Minimum length of the questions in question1 : " , min(data['q1_n_words']))

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

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

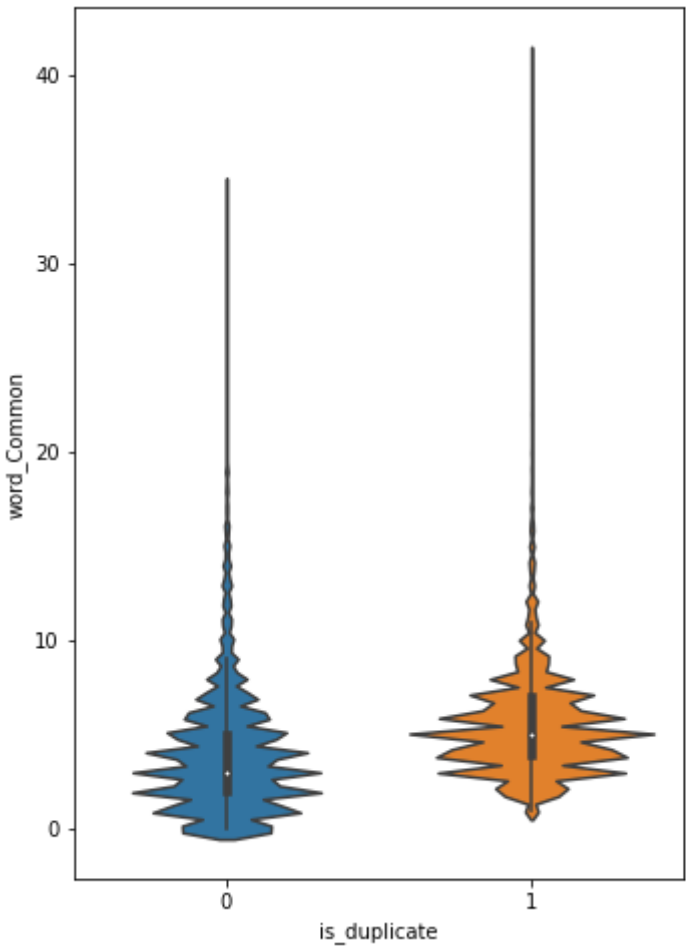
In [17]:
```
print ("maximum length of the questions in question1 : " , max(data['q1_n_words']))

print ("maximum length of the questions in question2 : " , max(data['q2_n_words']))

print ("Number of Questions with maximum length [question1] :", data[data['q1_n_words']== 125].shape[0])
print ("Number of Questions with maximum length [question2] :", data[data['q2_n_words']== 237].shape[0])
```

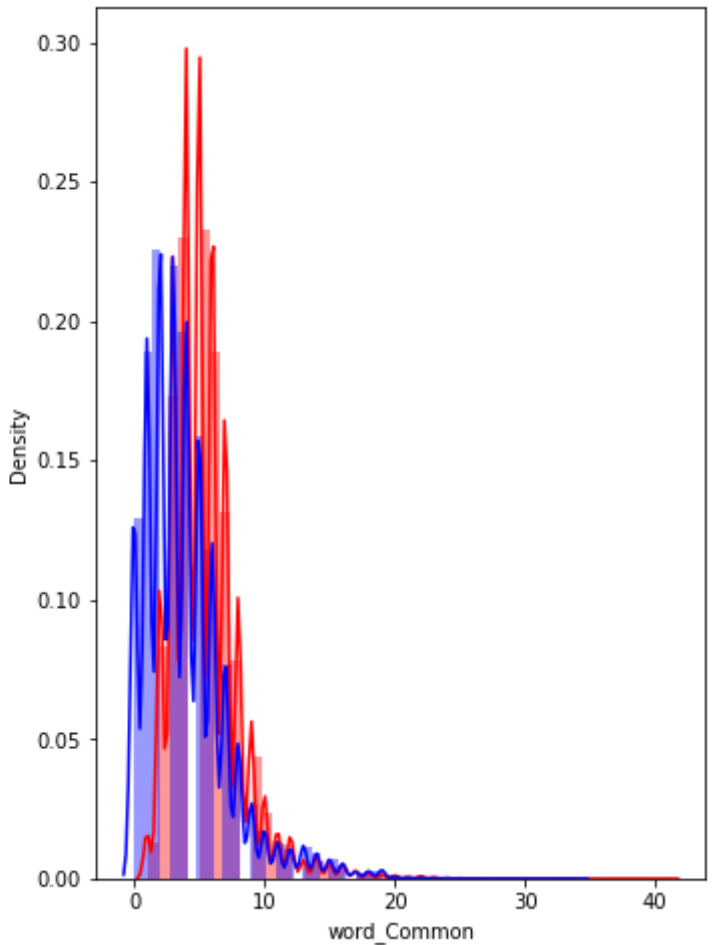
```
maximum length of the questions in question1 :  125
maximum length of the questions in question2 :  237
Number of Questions with maximum length [question1] : 1
Number of Questions with maximum length [question2] : 13
```

In [18]:
```
plt.figure(figsize=(12, 8))

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

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



In [19]:
```
plt.figure(figsize=(12, 8))

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

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





In [20]:
```
data=pd.read_csv('data_fe_without_preprocessing_train.csv')
```

In [21]:
```
data.shape
```

Out[21]:
```
(404290, 17)
```

In [22]:
```
data.head(2)
```

Out[22]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_sh
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	0.434
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	0.200

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

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 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 [23]:
```
# 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
```

In [24]:
```
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 [ ]:
```
data = extract_features(data)
```

In [ ]:
```
data.to_csv("nlp_features_train.csv", index=False)
```

In [34]:
```
data=pd.read_csv("nlp_features_train.csv")
```

In [38]:
```
data_duplicate = data[data['is_duplicate'] == 1]
datap_nonduplicate = data[data['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([data_duplicate["question1"], data_duplicate["question2"]]).flatten()
n = np.dstack([datap_nonduplicate["question1"], datap_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))
```

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

In [39]:
```
# 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 : 891339
Total number of words in non duplicate pair questions : 33193130
```

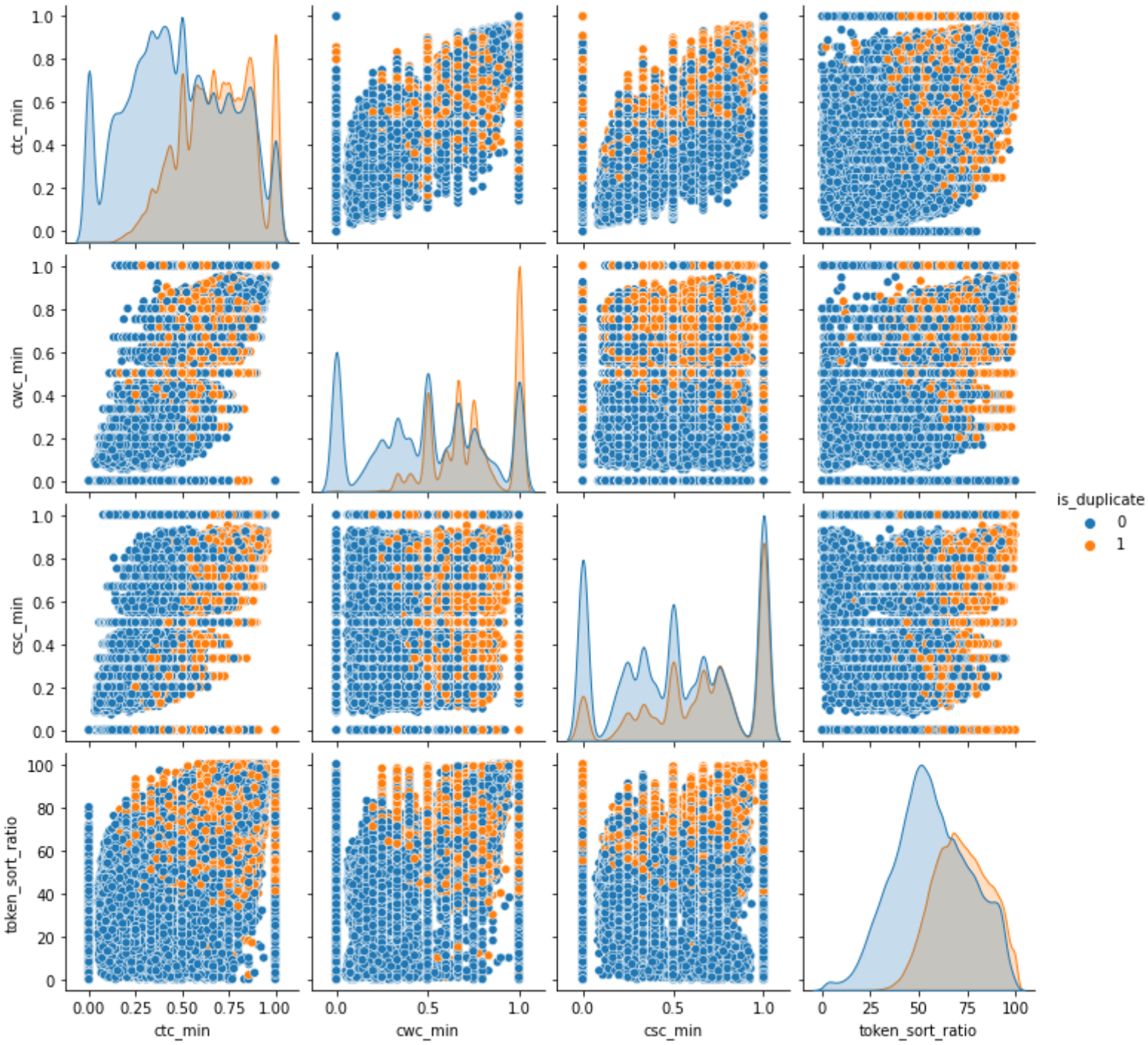
In [40]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

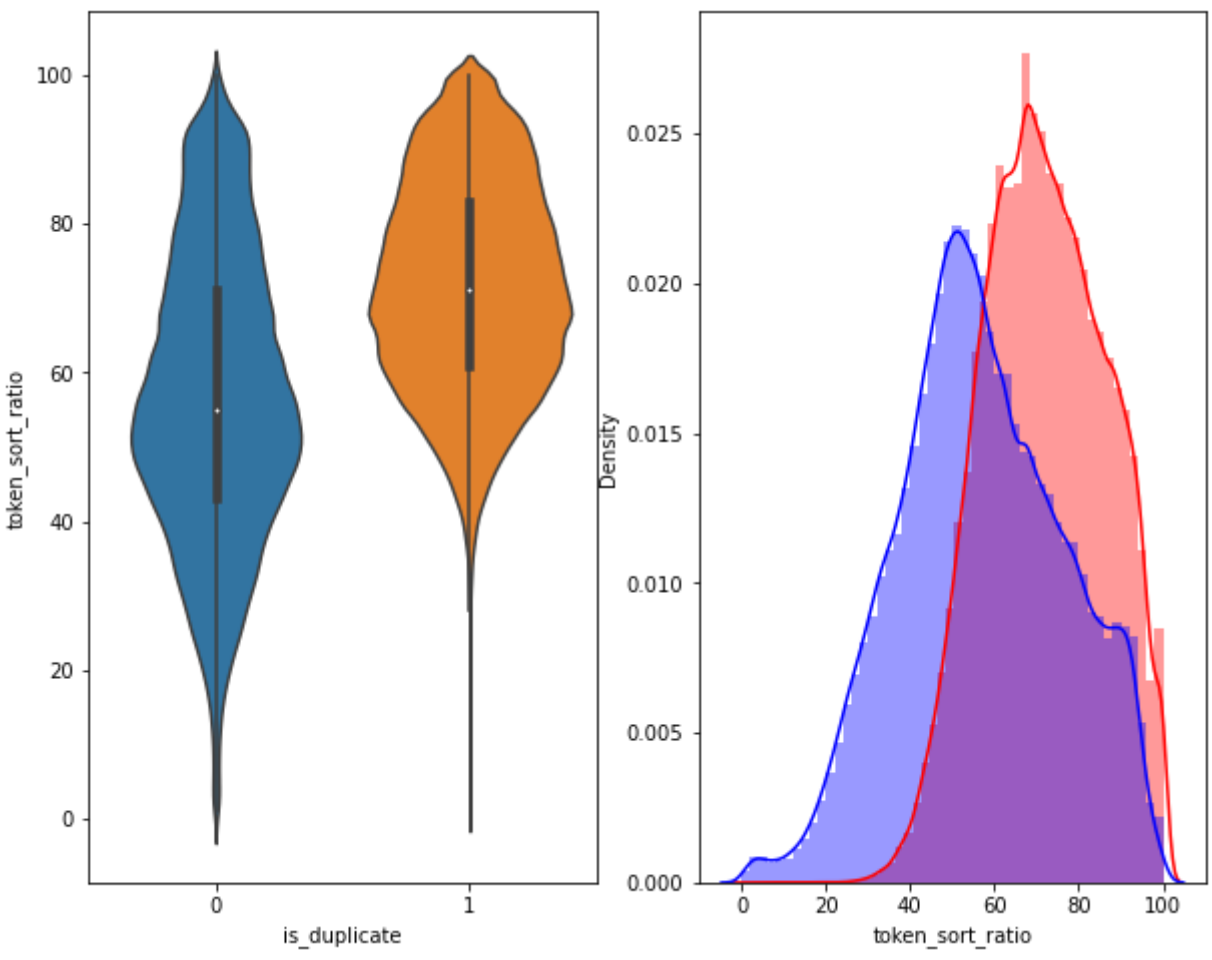
In [7]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

In [8]:
```
n = data.shape[0]
sns.pairplot(data[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min'
plt.show()
```

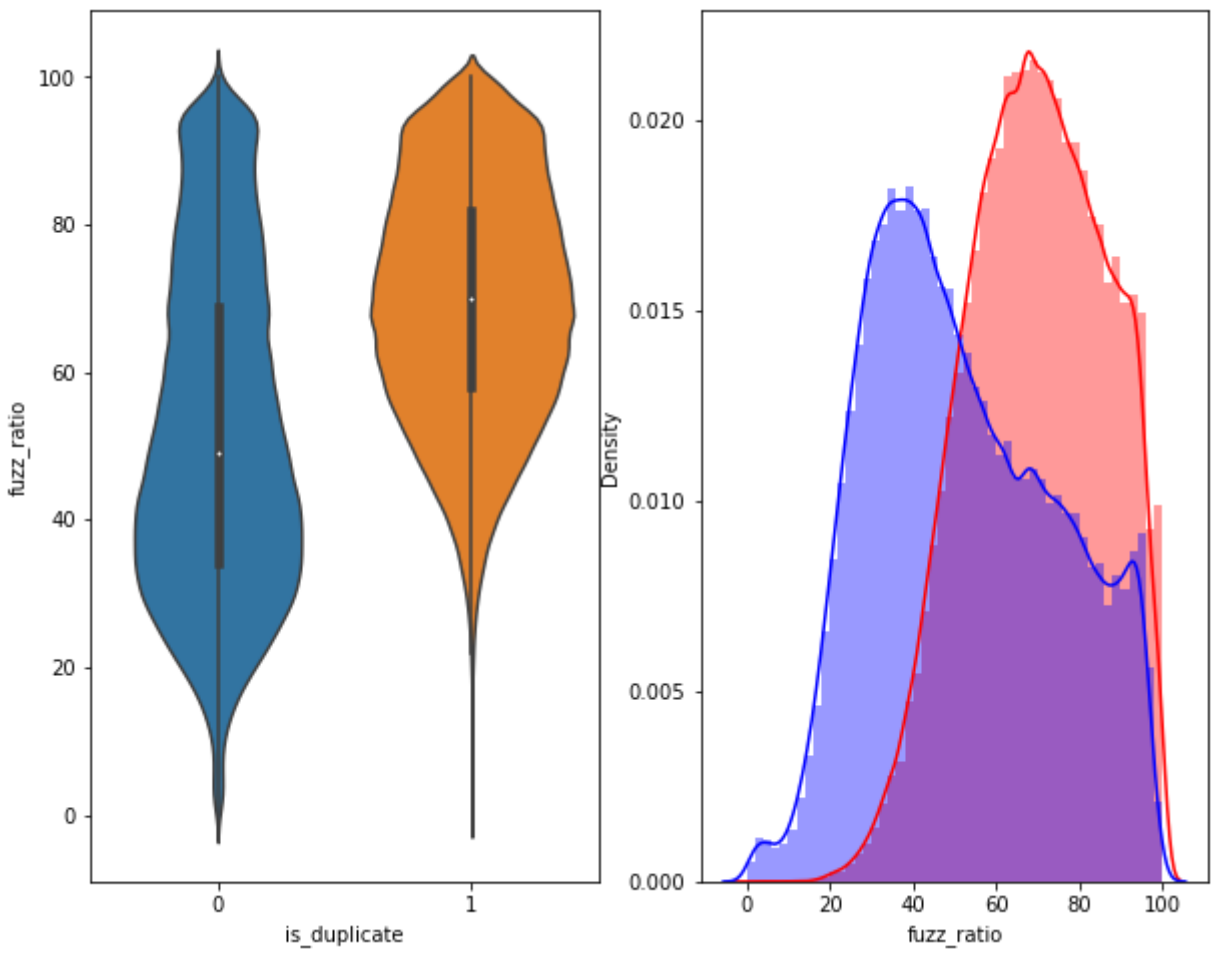


In [9]:
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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

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



In [10]:
```
plt.figure(figsize=(10, 8))

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

plt.subplot(1,2,2)
```

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



In [41]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

datap_subsampled = data[0:5000]
X = MinMaxScaler().fit_transform(datap_subsampled[['freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_C
y = datap_subsampled['is_duplicate'].values
```

In [44]:
```
data3=pd.DataFrame(X,columns=['freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Total',
```

In [45]:
```
data3.head(2)
```

Out[45]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	...	ctc_max	last_word_eq	fir
0	0.000000	0.0	0.176152	0.038894	0.200000	0.040000	0.40	0.10625	0.869565	0.000000	...	0.785714	0.0	
1	0.061224	0.0	0.135501	0.065687	0.107692	0.044444	0.16	0.08750	0.400000	0.025641	...	0.466667	0.0	

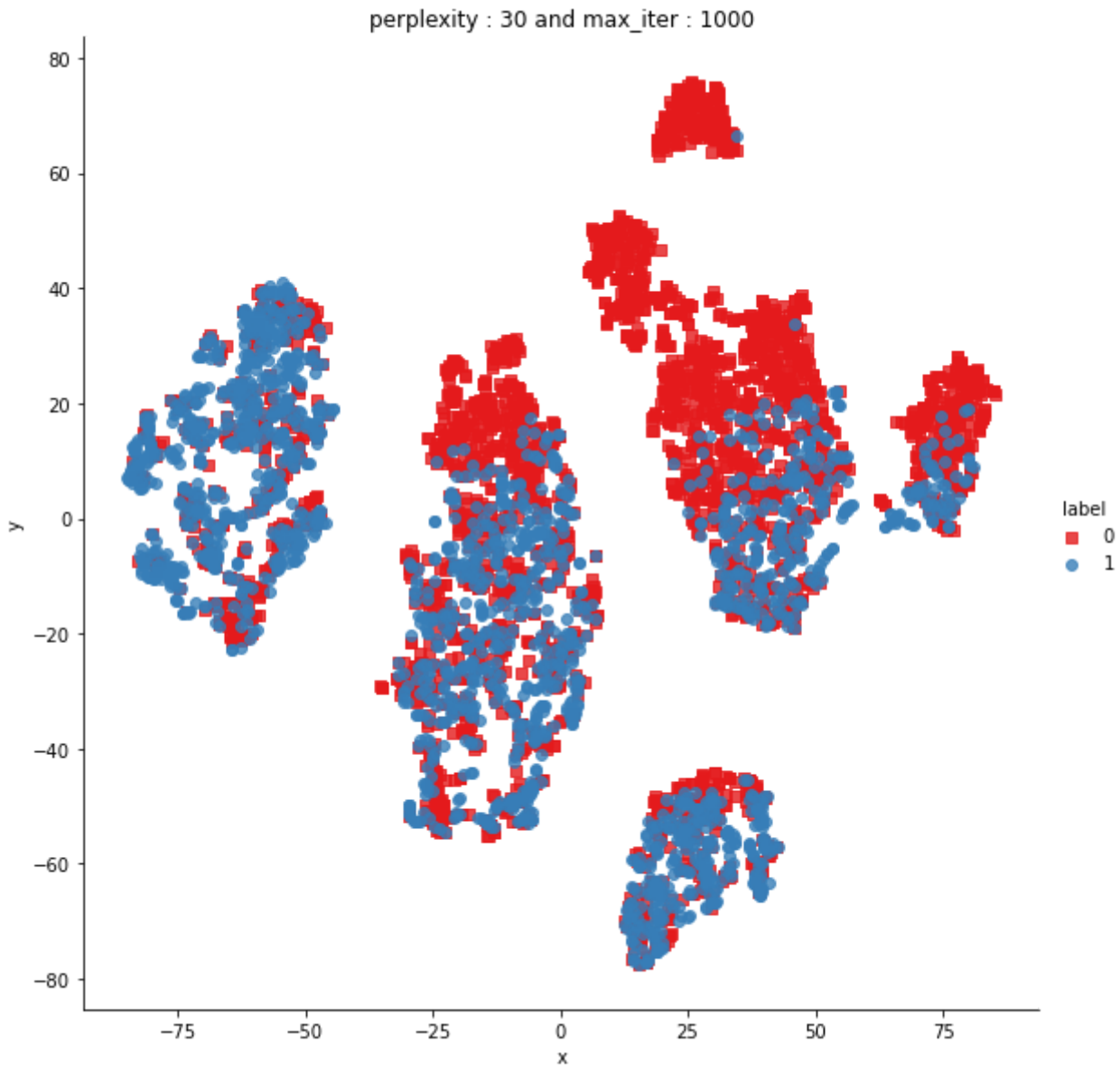
2 rows × 26 columns

In [14]:
```
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 0.001s...
[t-SNE] Computed neighbors for 5000 samples in 0.613s...
[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: 0.155586
[t-SNE] Computed conditional probabilities in 0.134s
[t-SNE] Iteration 50: error = 81.7986984, gradient norm = 0.0530740 (50 iterations in 1.064s)
[t-SNE] Iteration 100: error = 71.1705475, gradient norm = 0.0104491 (50 iterations in 0.859s)
[t-SNE] Iteration 150: error = 69.5886154, gradient norm = 0.0062177 (50 iterations in 0.800s)
[t-SNE] Iteration 200: error = 68.9021759, gradient norm = 0.0039960 (50 iterations in 0.824s)
[t-SNE] Iteration 250: error = 68.5032730, gradient norm = 0.0033806 (50 iterations in 0.778s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 68.503273
[t-SNE] Iteration 300: error = 1.8527544, gradient norm = 0.0011899 (50 iterations in 0.858s)
[t-SNE] Iteration 350: error = 1.4724221, gradient norm = 0.0004787 (50 iterations in 0.871s)
[t-SNE] Iteration 400: error = 1.3111446, gradient norm = 0.0002749 (50 iterations in 0.856s)
[t-SNE] Iteration 450: error = 1.2240133, gradient norm = 0.0001874 (50 iterations in 0.836s)
[t-SNE] Iteration 500: error = 1.1704545, gradient norm = 0.0001438 (50 iterations in 0.873s)
[t-SNE] Iteration 550: error = 1.1360605, gradient norm = 0.0001200 (50 iterations in 0.866s)
[t-SNE] Iteration 600: error = 1.1135497, gradient norm = 0.0001031 (50 iterations in 0.897s)
[t-SNE] Iteration 650: error = 1.0984259, gradient norm = 0.0000916 (50 iterations in 0.829s)
[t-SNE] Iteration 700: error = 1.0874386, gradient norm = 0.0000820 (50 iterations in 0.892s)
[t-SNE] Iteration 750: error = 1.0788362, gradient norm = 0.0000819 (50 iterations in 0.852s)
[t-SNE] Iteration 800: error = 1.0721704, gradient norm = 0.0000746 (50 iterations in 0.891s)
[t-SNE] Iteration 850: error = 1.0667583, gradient norm = 0.0000683 (50 iterations in 0.851s)
```

```
[t-SNE] Iteration 900: error = 1.0619664, gradient norm = 0.0000664 (50 iterations in 0.871s)
[t-SNE] Iteration 950: error = 1.0577466, gradient norm = 0.0000603 (50 iterations in 0.853s)
[t-SNE] Iteration 1000: error = 1.0538048, gradient norm = 0.0000595 (50 iterations in 0.859s)
[t-SNE] KL divergence after 1000 iterations: 1.053805
```

In [15]:
```
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 [16]:
```
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 0.000s...
[t-SNE] Computed neighbors for 5000 samples in 0.634s...
[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: 0.155586
[t-SNE] Computed conditional probabilities in 0.134s
[t-SNE] Iteration 50: error = 80.6523514, gradient norm = 0.0316805 (50 iterations in 1.685s)
[t-SNE] Iteration 100: error = 70.0516586, gradient norm = 0.0034348 (50 iterations in 1.024s)
[t-SNE] Iteration 150: error = 68.8465500, gradient norm = 0.0015942 (50 iterations in 0.926s)
[t-SNE] Iteration 200: error = 68.3692169, gradient norm = 0.0011727 (50 iterations in 0.986s)
[t-SNE] Iteration 250: error = 68.0969925, gradient norm = 0.0009325 (50 iterations in 0.920s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 68.096992
[t-SNE] Iteration 300: error = 1.5977951, gradient norm = 0.0007681 (50 iterations in 1.333s)
[t-SNE] Iteration 350: error = 1.2588005, gradient norm = 0.0002062 (50 iterations in 1.511s)
[t-SNE] Iteration 400: error = 1.1155961, gradient norm = 0.0001004 (50 iterations in 1.525s)
[t-SNE] Iteration 450: error = 1.0457357, gradient norm = 0.0000698 (50 iterations in 1.562s)
[t-SNE] Iteration 500: error = 1.0078565, gradient norm = 0.0000505 (50 iterations in 1.467s)
[t-SNE] Iteration 550: error = 0.9854025, gradient norm = 0.0000411 (50 iterations in 1.375s)
[t-SNE] Iteration 600: error = 0.9702448, gradient norm = 0.0000351 (50 iterations in 1.276s)
[t-SNE] Iteration 650: error = 0.9601687, gradient norm = 0.0000325 (50 iterations in 1.320s)
[t-SNE] Iteration 700: error = 0.9533580, gradient norm = 0.0000304 (50 iterations in 1.263s)
[t-SNE] Iteration 750: error = 0.9476262, gradient norm = 0.0000287 (50 iterations in 1.360s)
[t-SNE] Iteration 800: error = 0.9433101, gradient norm = 0.0000270 (50 iterations in 1.434s)
[t-SNE] Iteration 850: error = 0.9395683, gradient norm = 0.0000261 (50 iterations in 1.359s)
[t-SNE] Iteration 900: error = 0.9365825, gradient norm = 0.0000293 (50 iterations in 1.348s)
[t-SNE] Iteration 950: error = 0.9334615, gradient norm = 0.0000268 (50 iterations in 1.377s)
[t-SNE] Iteration 1000: error = 0.9303702, gradient norm = 0.0000238 (50 iterations in 1.279s)
[t-SNE] KL divergence after 1000 iterations: 0.930370
```

In [17]:
```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
```

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

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

In [ ]:

In [25]:
```
data3.head(2)
```

Out[25]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	...	ctc_max	last_word_eq	fir
0	0.000000	0.0	0.176152	0.038894	0.200000	0.040000	0.40	0.10625	0.869565	0.000000	...	0.785714	0.0	
1	0.061224	0.0	0.135501	0.065687	0.107692	0.044444	0.16	0.08750	0.400000	0.025641	...	0.466667	0.0	

2 rows × 26 columns

In [3]:
```
data1=pd.read_csv('train.csv')
```

In [4]:
```
data1.head(2)
```

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

In [5]:
```
data1['question1']=data1.question1.apply(lambda x: str(x))
data1['question2']=data1.question2.apply(lambda x: str(x))
```

In [6]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])

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

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

In [ ]:
```
# 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 (list(data1['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
data1['q1_feats_m'] = list(vecs1)
```

In [ ]:
```
vecs2 = []
for qu2 in (list(data1['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
data1['q2_feats_m'] = list(vecs2)
```

In [22]:
```
data1['q1_feats_m'] = list(vecs1)
data1['q2_feats_m'] = list(vecs2)
```

In [24]:
```
data1.head(2)
```

Out[24]:

	id	qid1	qid2	question1	question2	is_duplicate	q1_feats_m	q2_feats_m
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	[79.07837373018265, 15.782018959522247, 37.059...	[65.80132514238358, 15.163416892290115, 28.238...
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	[18.990903168916702, 48.39012713730335, 14.231...	[21.181431472301483, 44.1483453810215, -5.6844...

In [27]:
```
data2_q1 = pd.DataFrame(data1.q1_feats_m.values.tolist(), index= data1.index)
data2_q2 = pd.DataFrame(data1.q2_feats_m.values.tolist(), index= data1.index)
```

In [32]:
```
data2_q1.head(2)
```

Out[32]:

	0	1	2	3	4	5	6	7	8	9	...	87	88	89
0	79.078374	15.782019	37.059940	-28.544872	4.867482	16.195759	-23.889907	19.217555	45.637698	-44.844450	...	-34.502122	-37.652304	-24.22252
1	18.990903	48.390127	14.231475	-12.000782	-2.324469	-20.050934	-16.054571	-15.817222	3.254204	-39.863552	...	-34.506608	-42.934300	-23.45903

2 rows × 97 columns

In [31]:
```
data2_q2.head(2)
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	...	87	88	89

	0	1	2	3	4	5	6	7	8	9	...	87	88	89
0	65.801325	15.163417	28.238274	-22.443842	-1.128907	14.044980	-19.552754	23.298152	33.598356	-34.684082	...	-35.406901	-33.057641	-22.903015
1	21.181431	44.148345	-5.684423	-28.517999	-30.621333	7.486887	-16.820571	3.151194	11.878593	-13.489614	...	-31.066498	-45.401839	-23.039135

2 rows × 97 columns

In [30]:
```
data2_q1['id']=data1['id']
data2_q2['id']=data1['id']
```

In [46]:
```
data3['id']=data1['id']
```

In [47]:
```
data3.head(2)
```

Out[47]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	...	last_word_eq	first_word_eq
0	0.000000	0.0	0.176152	0.038894	0.200000	0.040000	0.40	0.10625	0.869565	0.000000	...	0.0	1.0
1	0.061224	0.0	0.135501	0.065687	0.107692	0.044444	0.16	0.08750	0.400000	0.025641	...	0.0	1.0

2 rows × 27 columns

In [48]:
```
ques  = data2_q1.merge(data2_q2, on='id',how='left')
```

In [50]:
```
result=data3.merge(ques,on='id',how='left')
```

In [57]:
```
result['is_duplicate']=data1['is_duplicate']
```

In [59]:
```
result.to_csv('final_features1.csv')
```

In [44]:
```
data4=pd.read_csv('final_features1.csv')
```

In [45]:
```
data4.head(2)
```

Out[45]:

	Unnamed: 0	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	...	87_y	88_y	
0	0	0.000000	0.0	0.176152	0.038894	0.200000	0.040000	0.40	0.10625	0.869565	...	-35.406901	-33.057641	-22.
1	1	0.061224	0.0	0.135501	0.065687	0.107692	0.044444	0.16	0.08750	0.400000	...	-31.066498	-45.401839	-23.

2 rows × 221 columns

In [ ]:
```

```

In [46]:
```
data4.columns
```

Out[46]:
```
Index(['Unnamed: 0', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share',
       ...
       '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y',
       'is_duplicate'],
      dtype='object', length=221)
```

In [47]:
```
y_true = data4['is_duplicate']
data4.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace=True)
```

In [48]:
```
data4.head(3)
```

Out[48]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	...	86_y	87_y	
0	0.000000	0.0	0.176152	0.038894	0.200000	0.040000	0.40	0.10625	0.869565	0.000000	...	-13.240540	-35.406901	-3:
1	0.061224	0.0	0.135501	0.065687	0.107692	0.044444	0.16	0.08750	0.400000	0.025641	...	-37.242324	-31.066498	-4!
2	0.000000	0.0	0.195122	0.040622	0.200000	0.031111	0.16	0.11250	0.333333	0.000000	...	-47.572795	-25.605735	-:

3 rows × 218 columns

In [49]:
```
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 train_test_split
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.metrics import log_loss

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 [ ]:

In [50]:
```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train, y_test=train_test_split(data4, y_true, test_size=0.33,stratify = y_true)
```

In [51]:
```
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 : (3350, 218)
Number of data points in test data : (1650, 218)
```

In [52]:
```
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:  0.6179104477611941 Class 1:  0.382089552238806
---------- Distribution of output variable in train data ----------
Class 0:  0.38242424242424244 Class 1:  0.38242424242424244
```

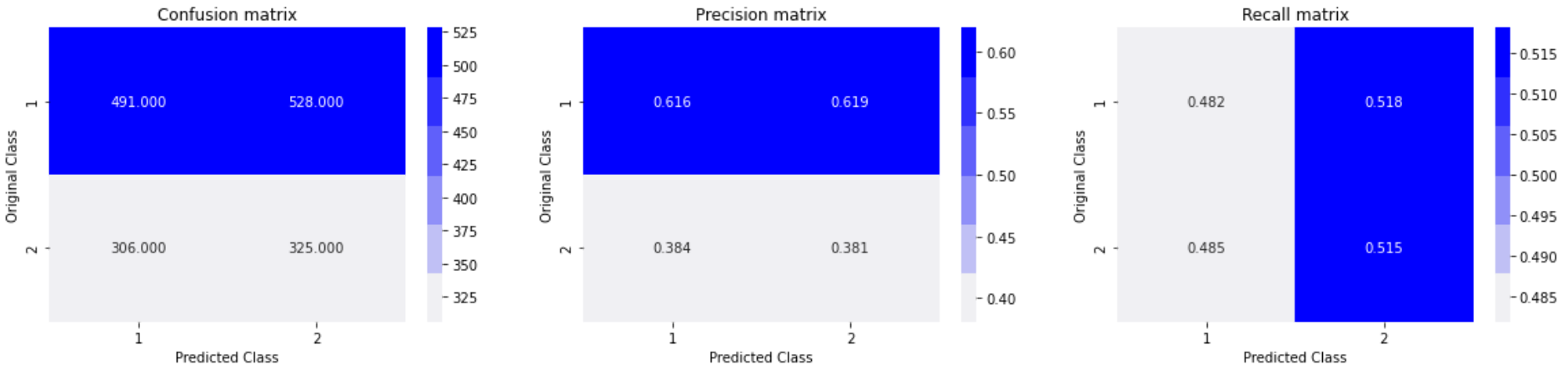
In [53]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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()
```

In [54]:

```
# 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 0.8994702212712212
```

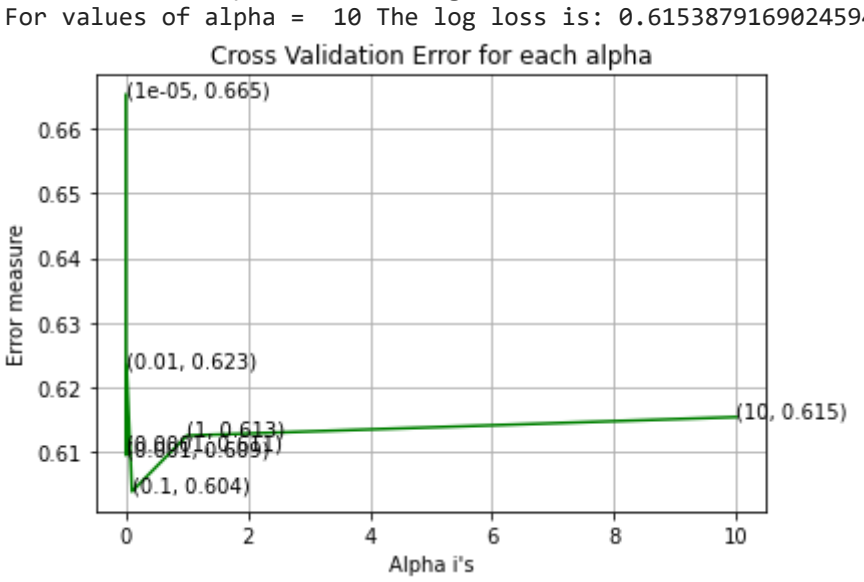


In [55]:

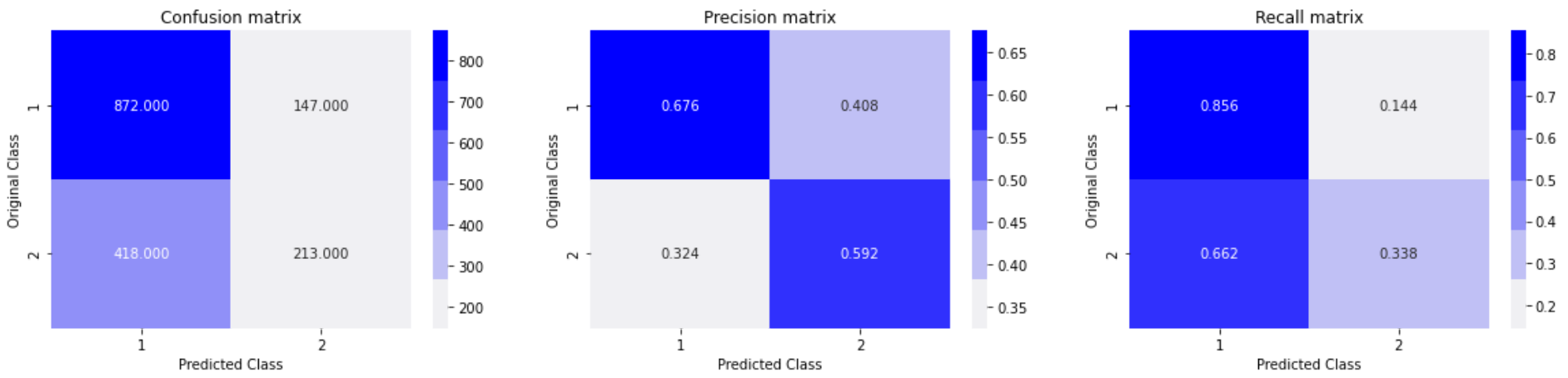
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, 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.classes_,
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, e
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: 0.6652384485934735
For values of alpha =  0.0001 The log loss is: 0.6106260468690239
For values of alpha =  0.001 The log loss is: 0.609483534793384
For values of alpha =  0.01 The log loss is: 0.6233318963497547
For values of alpha =  0.1 The log loss is: 0.6038755250259339
For values of alpha =  1 The log loss is: 0.612519518298032
For values of alpha =  10 The log loss is: 0.6153879169024594
```




```
For values of best alpha =  0.1 The train log loss is: 0.5726219718062434
For values of best alpha =  0.1 The test log loss is: 0.6038755250259339
Total number of data points : 1650
```

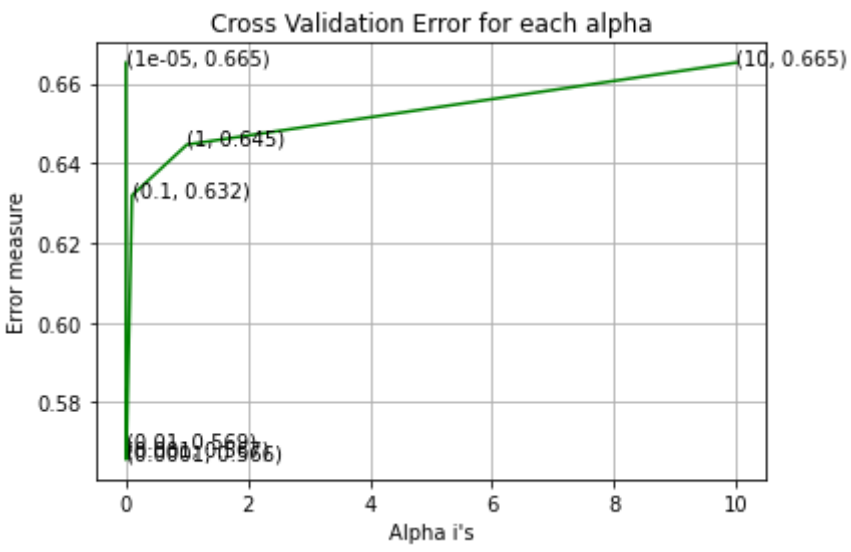


In [56]:

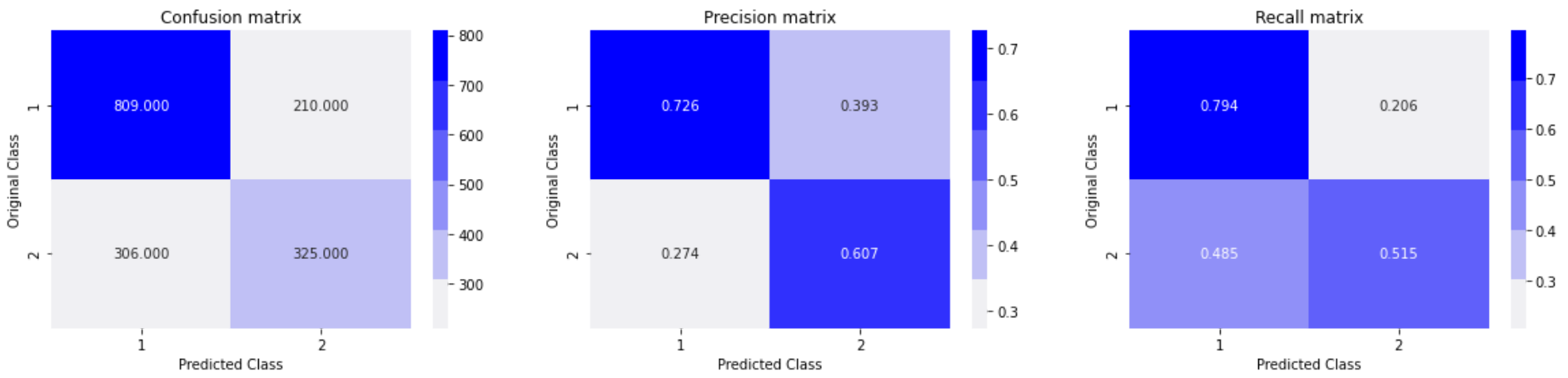
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, 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.classes_,
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, e
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: 0.6652384485934735
For values of alpha =  0.0001 The log loss is: 0.5655997521438436
For values of alpha =  0.001 The log loss is: 0.5667947898146576
For values of alpha =  0.01 The log loss is: 0.5685077509777603
For values of alpha =  0.1 The log loss is: 0.6319035169423035
For values of alpha =  1 The log loss is: 0.6446746005470357
For values of alpha =  10 The log loss is: 0.6652386790480342
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5349601632449692
For values of best alpha =  0.0001 The test log loss is: 0.5655997521438436
Total number of data points : 1650
```



In [64]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
#from sklearn.ensemble import RandomForestClassifier
import time
start_time = time.time()
parameters={ 'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3] ,'min_samples_split':[5, 10, 100],'n_estimators':[5,10,50, 75, 1
gbdt = XGBClassifier(eval_metric='logloss',verbosity=0)
clf=RandomizedSearchCV(gbdt,parameters,scoring='roc_auc',return_train_score=True,verbose=10,n_jobs=-1)
clf.fit(X_train, y_train)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
--- 59.846250772476196 seconds ---
```

In [66]:
```
print(clf.best_score_)
print(clf.best_estimator_)
print(clf.best_params_)
```

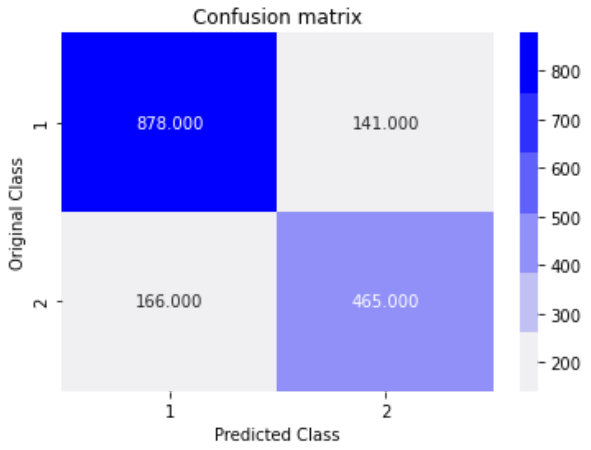
```
0.8978223128019325
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.2, max_delta_step=0,
              max_depth=6, min_child_weight=1, min_samples_split=5, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=0)
{'n_estimators': 100, 'min_samples_split': 5, 'learning_rate': 0.2}
```

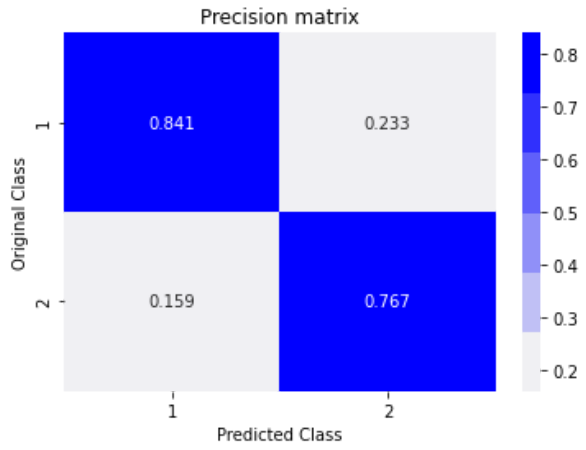
In [67]:
```
print('Best score: ',clf.best_score_)
#print('k value with best score: 'clf.best_params_)
#print(clf.cv_results_) print(clf.best_params_)
learn=clf.best_params_['learning_rate']
split=clf.best_params_['min_samples_split']
esti=clf.best_params_['n_estimators']
```

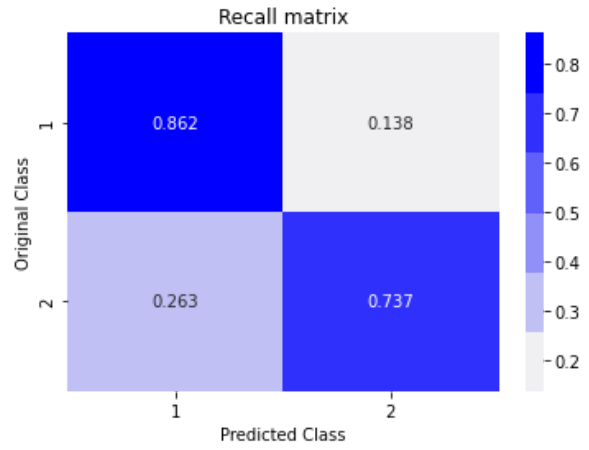
```
Best score:  0.8978223128019325
```

In [68]:
```
model = XGBClassifier(eta=learn,min_samples_split=split,n_estimators=esti,eval_metric='logloss',verbosity=0)
model.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(model, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_,
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, e
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 =  0.0001 The train log loss is: 0.15006075896707094
For values of best alpha =  0.0001 The test log loss is: 0.40067144284698575
Total number of data points : 1650
```







In [69]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "log-loss"]
x.add_row(["TFIDF  W2V", "Logistic_Regression" , 0.60])
x.add_row(["TFIDF W2V", "Linear_SVM" , 0.56])
x.add_row(["TFIDF W2V", "XGBoost" , 0.40])
print(x)
```

```
+------------+---------------------+----------+
| Vectorizer |        Model        | log-loss |
+------------+---------------------+----------+
| TFIDF  W2V | Logistic_Regression |   0.6    |
| TFIDF W2V  |      Linear_SVM     |   0.56   |
| TFIDF W2V  |       XGBoost       |   0.4    |
+------------+---------------------+----------+
```

In [ ]: