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?","W
hat 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 happe
n 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 geol
ogist?","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 re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two stri
ngs
# 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-pyt
hon3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image

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

```
C:\Users\Roshan\Anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarnin
g:

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

3.1 Reading data and basic stats

In [21]:
```
df = pd.read_csv("train.csv")

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

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

In [22]:
```
df.head()
```

Out[22]:

	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 [23]:
```
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:

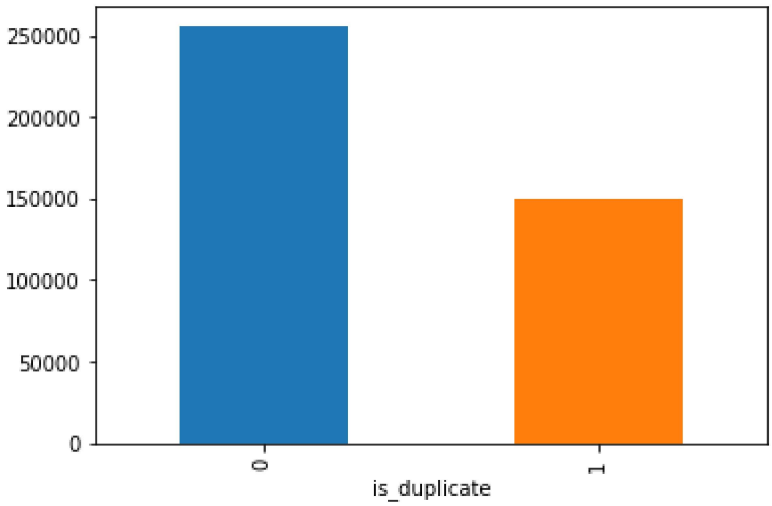
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

In [24]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x200ad0405c0>
```



In [25]:
```
print('~> Total number of question pairs for training:\n   {}'.format(len(df
)))
```

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

In [26]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(1
00 - round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(rou
nd(df['is_duplicate'].mean()*100, 2)))
```

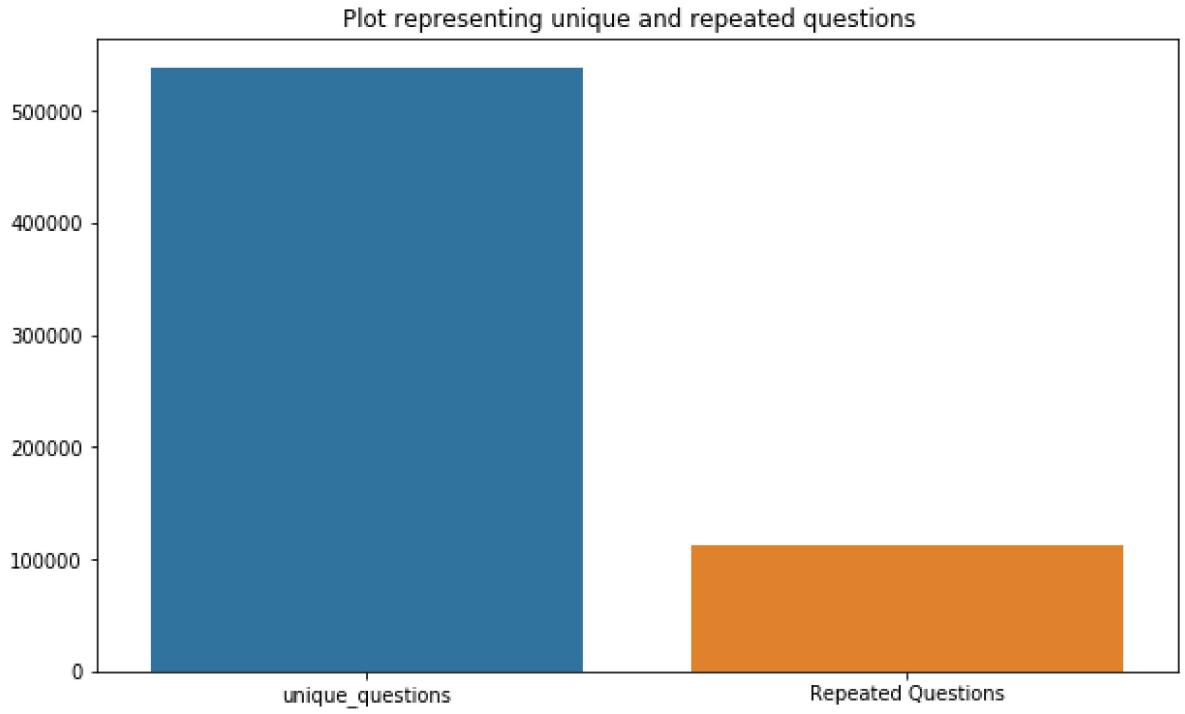
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.2 Number of unique questions

In [27]:
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

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

print ('Max number of times a single question is repeated: {}\n'.format(max(qi
ds.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 (20.7795394
5937505%)

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

In [28]:
```
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

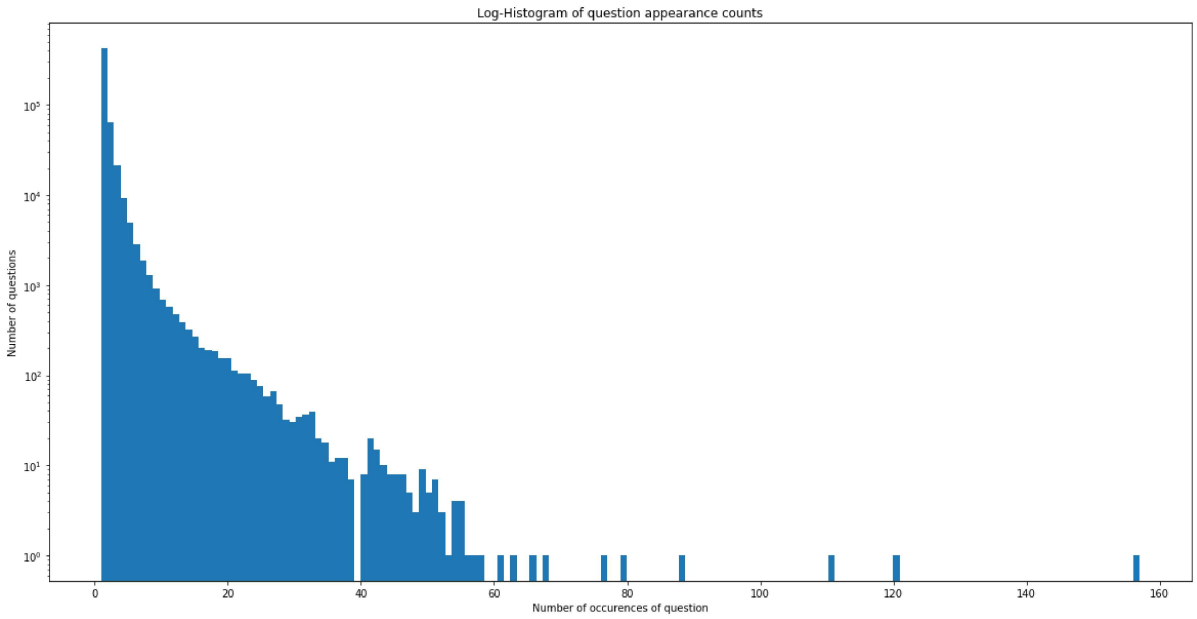
In [29]:
```
#checking whether there are any repeated pair of questions

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 [30]:
```
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(ma
x(qids.value_counts())))
```

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



3.2.5 Checking for NULL values

In [31]:
```
#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 [32]:
```
# 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 [33]:
```
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[33]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_
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	
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	
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	
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	
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	

3.3.1 Analysis of some of the extracted features

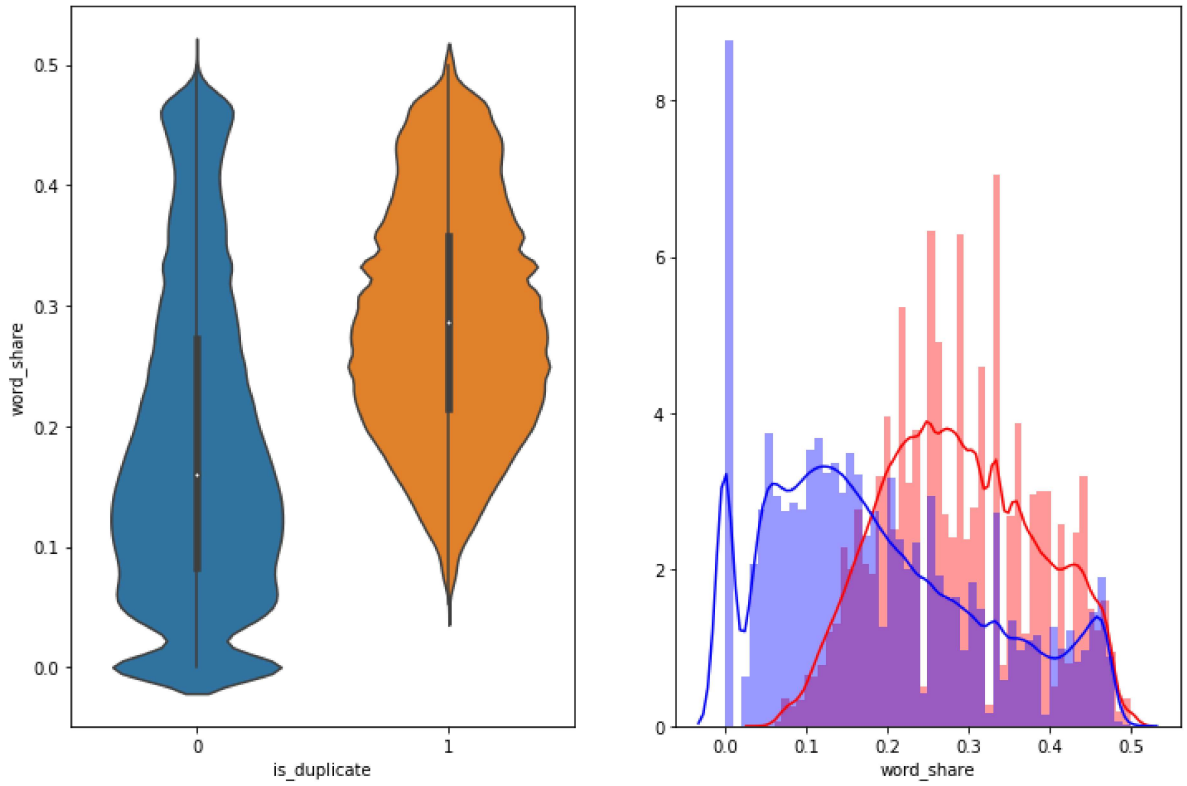
- Here are some questions have only one single words.

In [34]:
```
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_wo
rds']== 1].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_wo
rds']== 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

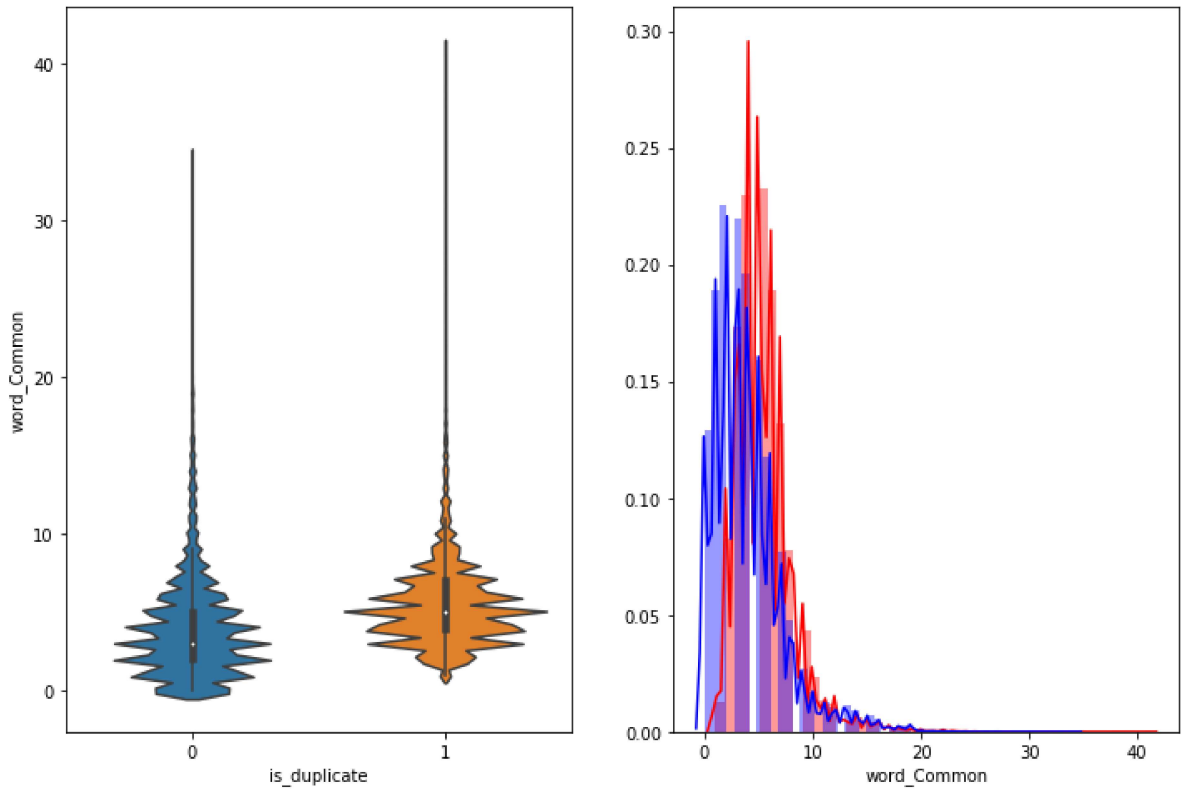
In [35]:
```
plt.figure(figsize=(12, 8))

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", co
lor = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , c
olor = '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 [36]:
```
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", c
olor = '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

1.2.1 : EDA: Advanced Feature Extraction.

In [5]:
```
#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 [6]:
```
df.head(2)
```

Out[6]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_
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	
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	

3.4 Preprocessing of Text

In [7]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Roshan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[7]: True

In [8]:
```
# 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("’", "'").repl
ace("’", "'")\
                           .replace("won't", "will not").replace("cannot", "ca
n 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").rep
lace("'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 [9]:
```
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_token
s)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_token
s)) + 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-func
tion-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["q
uestion1"], x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sort
ing the tokens alphabetically, and
    # then joining them back into a string We then compare the transformed str
ings 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["que
stion1"], 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 [10]:
```
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[10]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
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
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

2 rows × 21 columns

3.5.1 Analysis of extracted features

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

In [12]:
```
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}} t
o {1,2,3,4}
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten
()
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).
flatten()

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

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

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

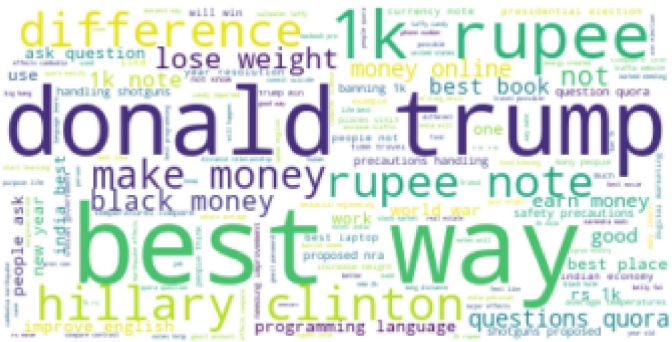
In [13]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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

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

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

Word Clouds generated from duplicate pair question's text

In [14]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=sto
pwords)
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
```



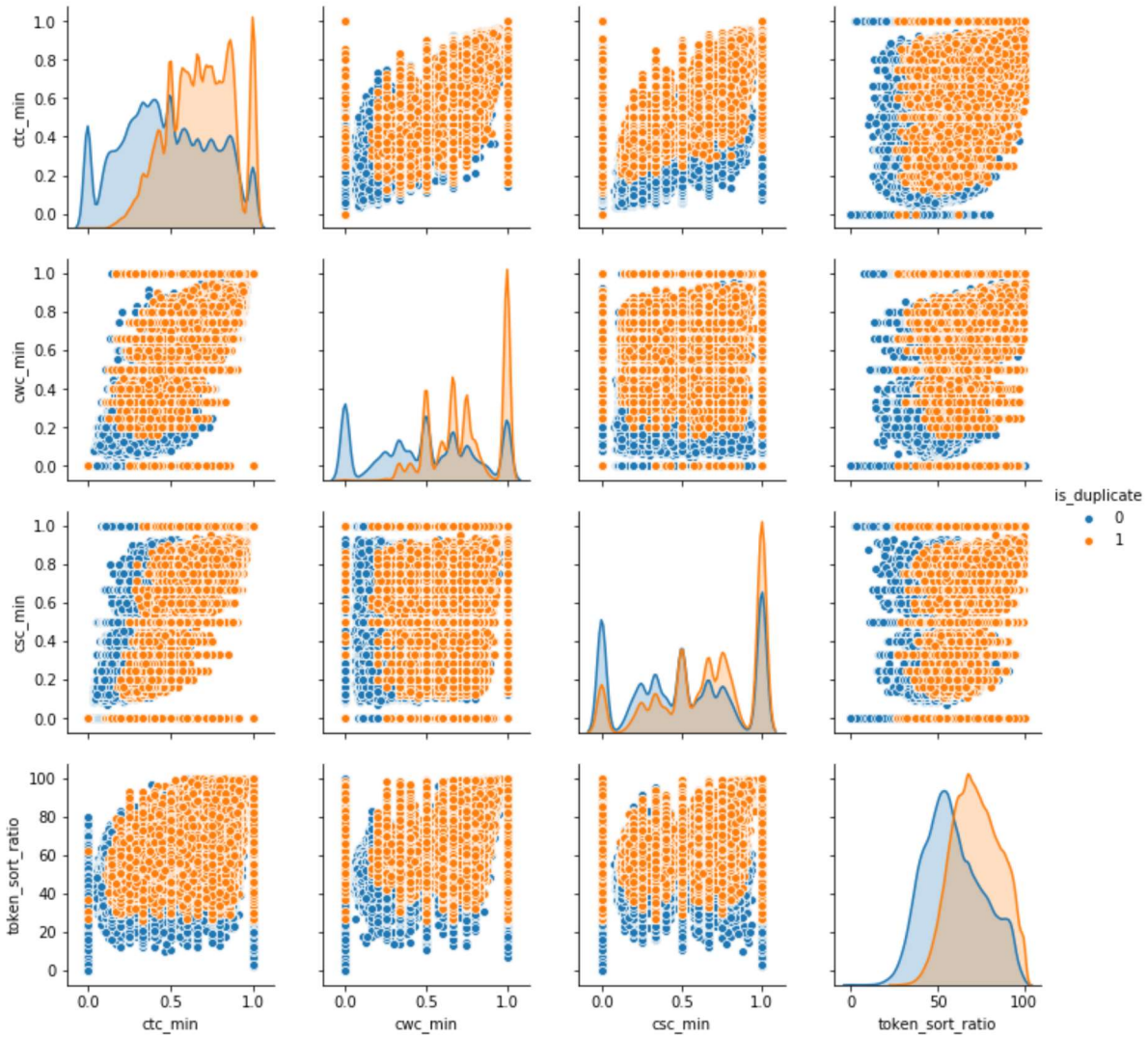
Word Clouds generated from non duplicate pair question's text

In [15]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stop
words)
# 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()
```

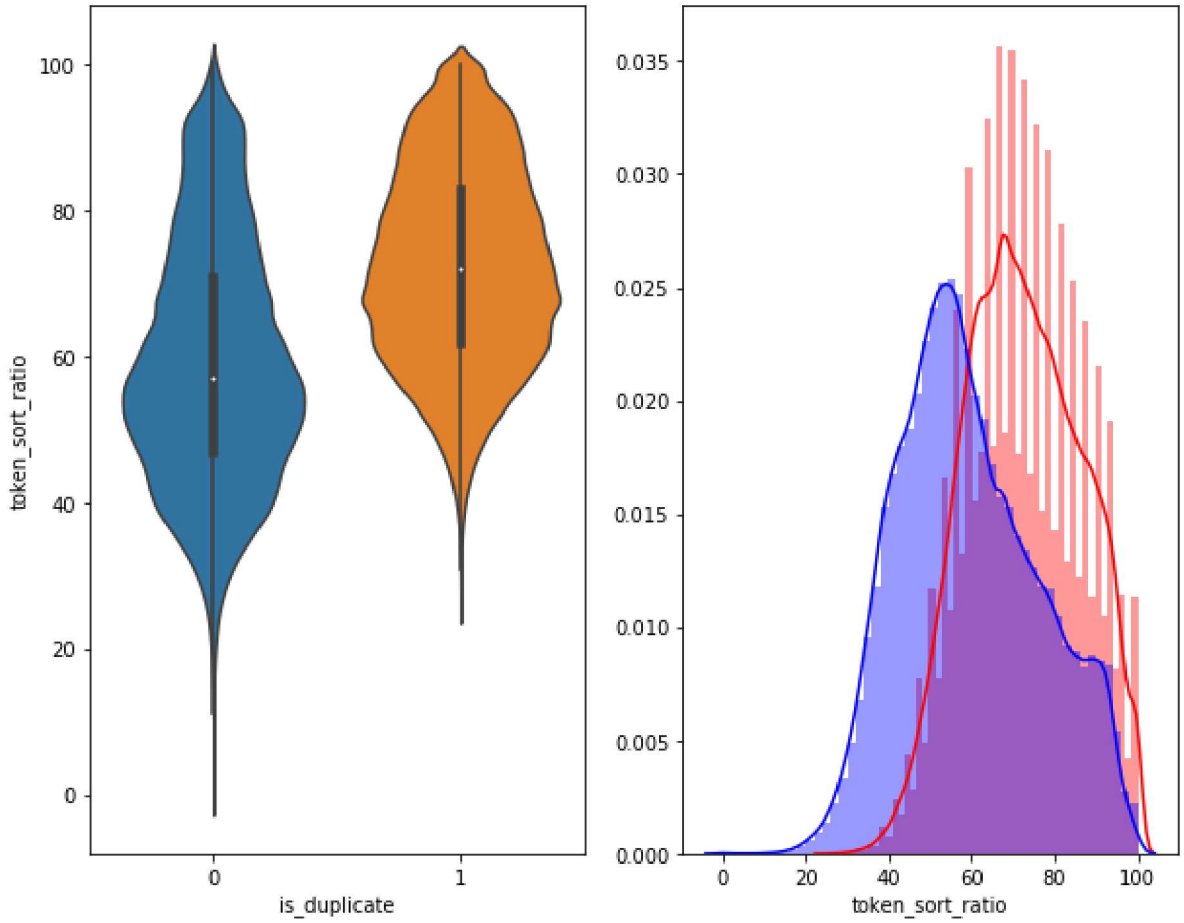
```
Word Cloud for non-Duplicate Question pairs:
```

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

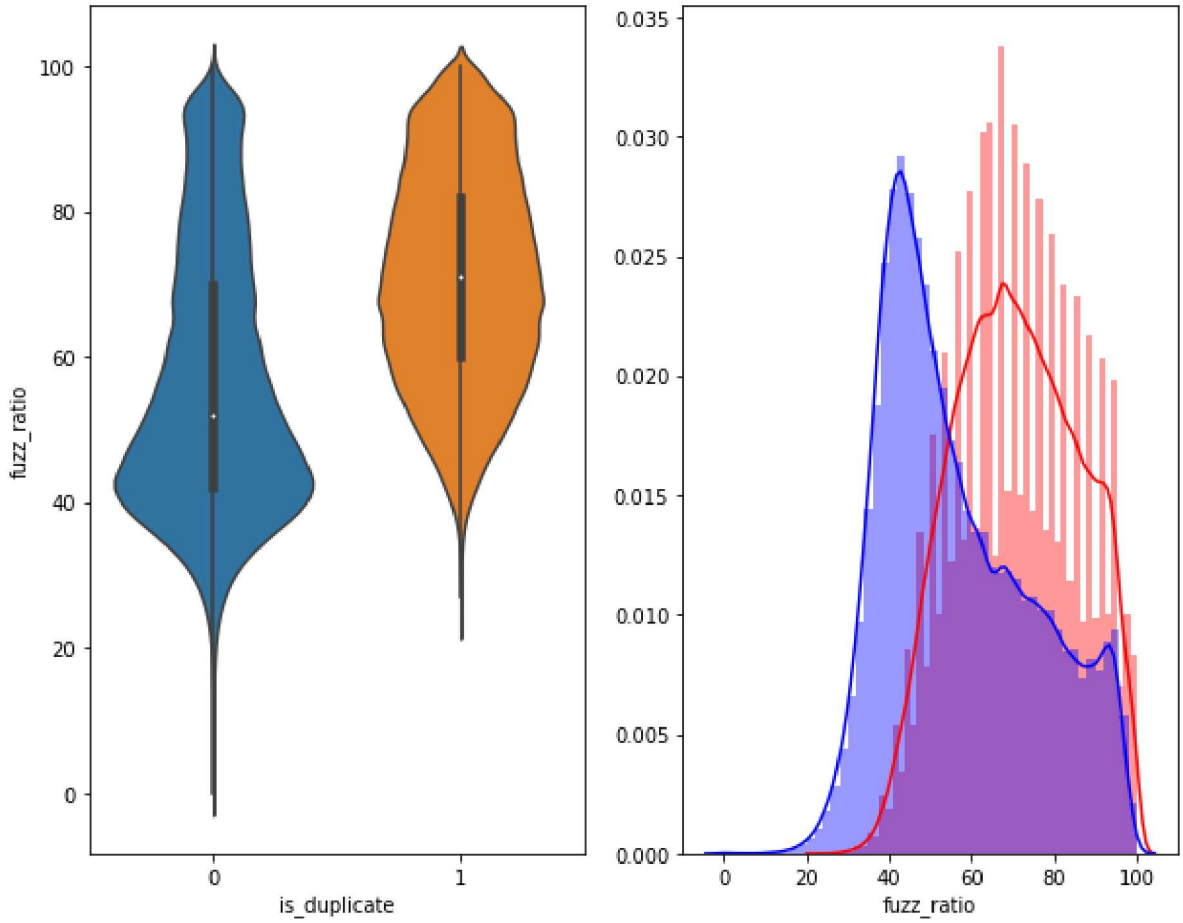
In [16]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```



In [17]:
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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

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



In [18]:
```
plt.figure(figsize=(10, 8))

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

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



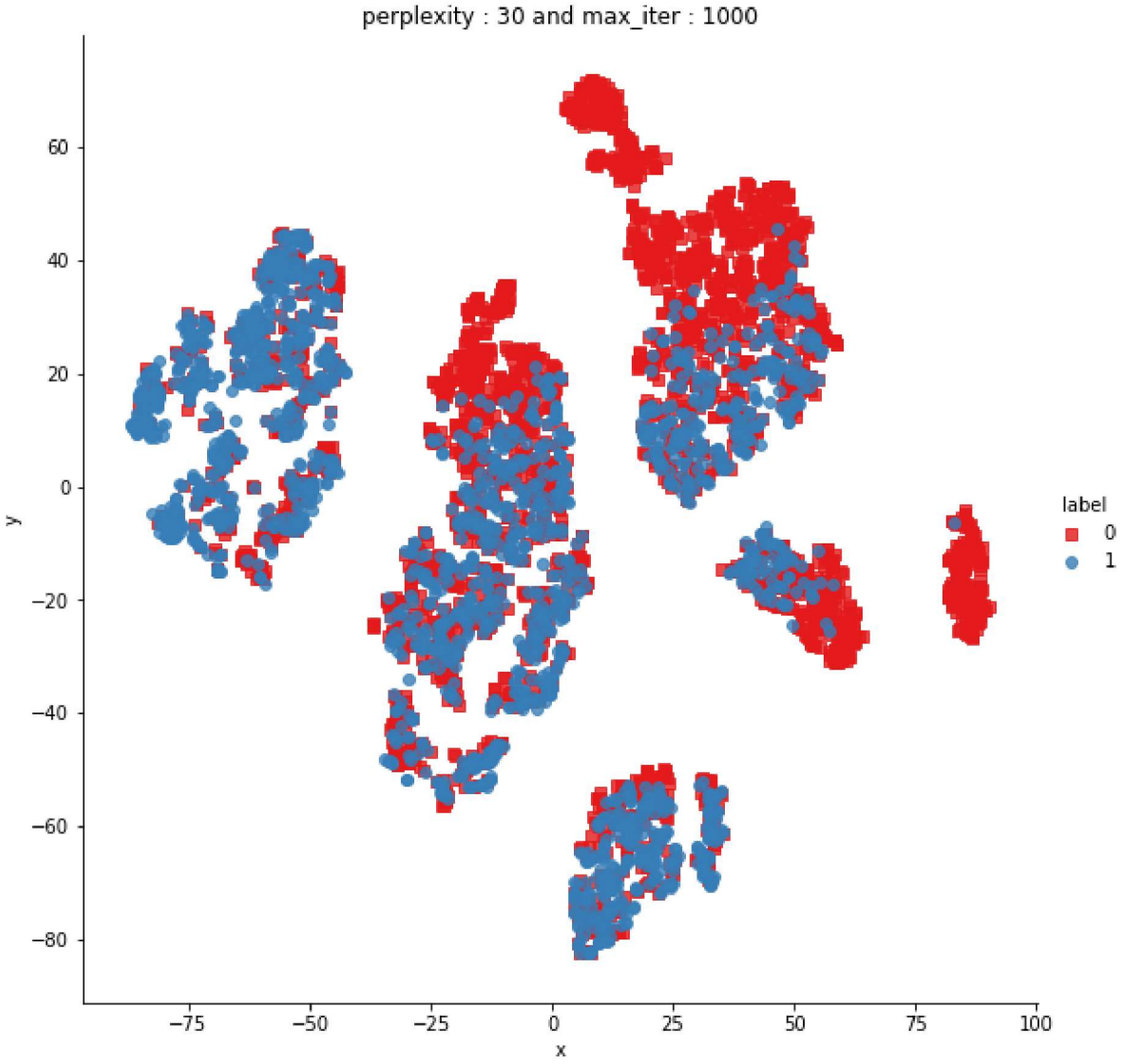
3.5.2 Visualization

In [19]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after clea
ning 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_mi
n', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_word_eq' , 'abs
_len_diff' , 'mean_len' , 'token_set_ratio' , 'token_sort_ratio' ,  'fuzz_ratio' , 'fuzz_partial_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

In [20]:
```
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 0.130s...
[t-SNE] Computed neighbors for 5000 samples in 0.690s...
[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 0.519s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 itera
tions in 6.128s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iter
ations in 4.474s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iter
ations in 4.425s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iter
ations in 4.661s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iter
ations in 4.801s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.273346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 itera
tions in 4.728s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 itera
tions in 4.482s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 itera
tions in 4.525s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 itera
tions in 4.529s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 itera
tions in 4.541s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 itera
tions in 4.585s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 itera
tions in 4.521s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 itera
tions in 4.715s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 itera
tions in 4.730s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 itera
tions in 4.654s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 itera
tions in 4.686s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 itera
tions in 4.612s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 itera
tions in 4.603s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 itera
tions in 4.678s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iter
ations in 4.698s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [21]:
```
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()
```

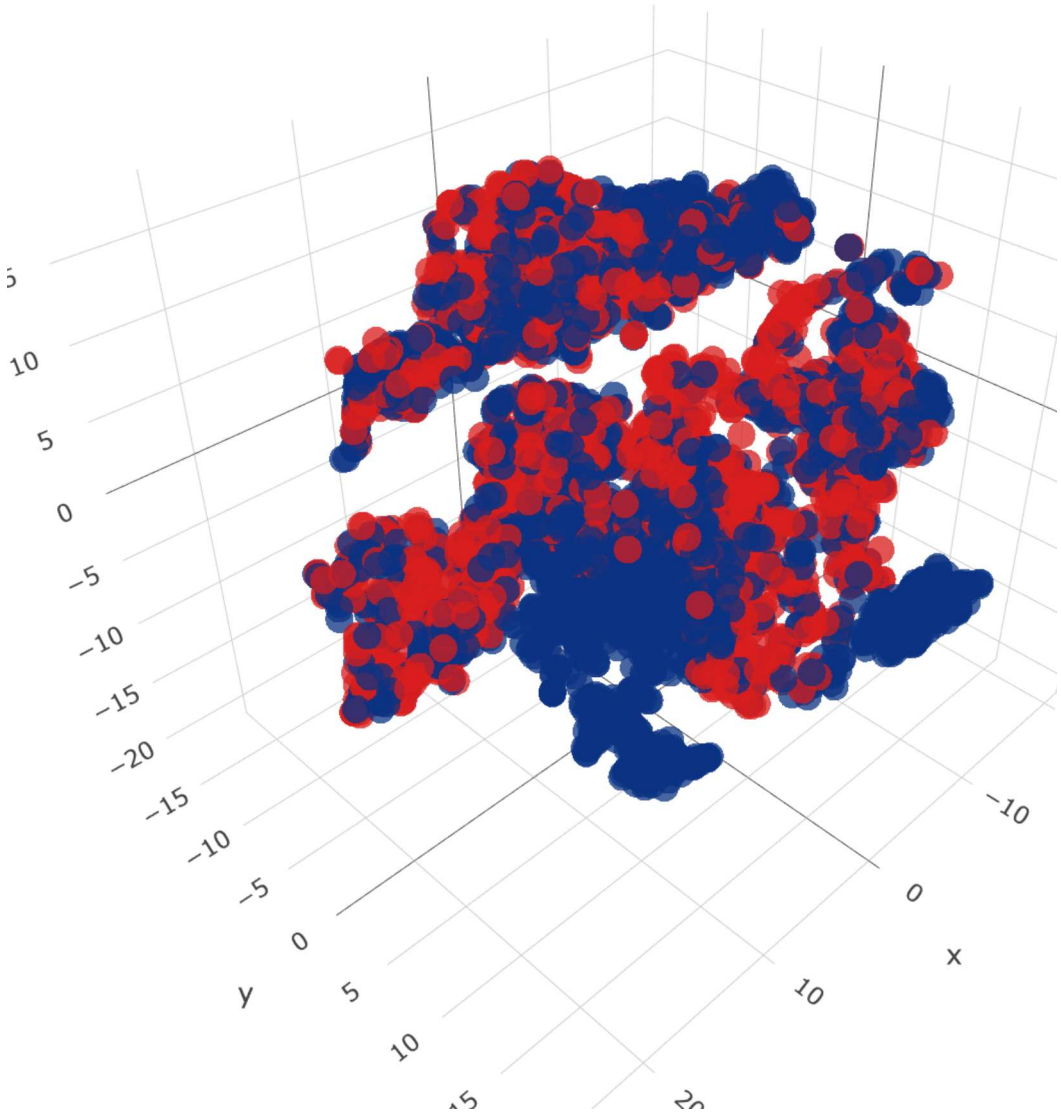


In [22]:
```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.024s...
[t-SNE] Computed neighbors for 5000 samples in 0.650s...
[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 0.444s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 itera
tions in 18.156s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (50 iter
ations in 10.178s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (50 iter
ations in 9.169s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (50 iter
ations in 9.164s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (50 iter
ations in 8.936s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729782
[t-SNE] Iteration 300: error = 1.4978341, gradient norm = 0.0006938 (50 itera
tions in 14.754s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50 itera
tions in 15.885s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50 itera
tions in 15.092s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50 itera
tions in 15.017s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50 itera
tions in 14.677s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50 itera
tions in 14.791s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50 itera
tions in 14.484s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50 itera
tions in 14.601s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50 itera
tions in 14.673s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50 itera
tions in 14.675s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50 itera
tions in 15.216s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50 itera
tions in 14.652s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50 itera
tions in 14.484s)
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50 itera
tions in 14.829s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (50 iter
ations in 14.667s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [23]:
```
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 feature
s')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

3d embedding with engineered features



3.6 Featurizing text data with tfidf weighted word-vectors

In [2]:
```
# 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 [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]:
```
#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",nrows=100000,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",nrows=100000,encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [5]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = dfnlp[['id','question1','question2']]
duplicate = dfnlp.is_duplicate
```

In [6]:
```
df3 = df3.fillna(' ')
#assigning new dataframe with columns question(q1+q2) and id same as df3
new_df = pd.DataFrame()
new_df['questions'] = df3.question1 + ' ' + df3.question2
new_df['id'] = df3.id
df2['id']=df1['id']
new_df['id']=df1['id']
final_df = df1.merge(df2, on='id',how='left') #merging df1 and df2
X  = final_df.merge(new_df, on='id',how='left')#merging final_df and new_df
```

In [7]:
```
X.columns
```

Out[7]:
```
Index(['id', 'is_duplicate', 'cwc_min', 'cwc_max', 'csc_min', 'csc_max',
       'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq', 'abs_len_diff',
       'mean_len', 'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [9]:
```
#removing id from X
X=X.drop(['id','is_duplicate'],axis=1)
X.columns
```

Out[9]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [10]:
```
target = np.array(duplicate)
#df.drop(['id','is_duplicate'], axis=1, inplace=True)
```

Split Data into Train and Test Data in the ratio of 70:30

In [12]:
```
from sklearn.model_selection import train_test_split

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

In [13]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
(70000, 27) (70000,)
(30000, 27) (30000,)
====================================================================================================
```

In [14]:
```
#seperating questions for tfidf vectorizer
X_train_ques=X_train['questions']
X_test_ques=X_test['questions']

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

Applying TFIDF W2V Vectorizer

In [15]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# list_of_sentance_train=[]
# for sentance in X_train_ques:
#     list_of_sentance_train.append(sentance.split())

tfidf = TfidfVectorizer(lowercase=False )
tfidf.fit_transform(X_train_ques)

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

In [16]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
import en_core_web_sm

nlp = en_core_web_sm.load()
```

In [19]:
```
from tqdm import tqdm

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_train_ques)):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
#df['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 70000/70000 [32:03<00:00, 33.19it/s]
```

In [20]:
```
vecs2 = []
for qu2 in tqdm(list(X_test_ques)):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 96])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
#df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 30000/30000 [13:53<00:00, 33.08it/s]
```

In [21]:
```
first_df=pd.DataFrame(vecs1)
sec_df=pd.DataFrame(vecs2)
```

In [22]:
```
from scipy.sparse import hstack
X_train = hstack((X_train.values,first_df))
X_test= hstack((X_test.values,sec_df))
print(X_train.shape)
print(X_test.shape)
```

```
(70000, 122)
(30000, 122)
```

4. Machine Learning Models

In [23]:
```
print("Number of data points in train data :",X_train.shape)
print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (70000, 122)
Number of data points in test data : (30000, 122)
```

In [24]:
```
from collections import Counter

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

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

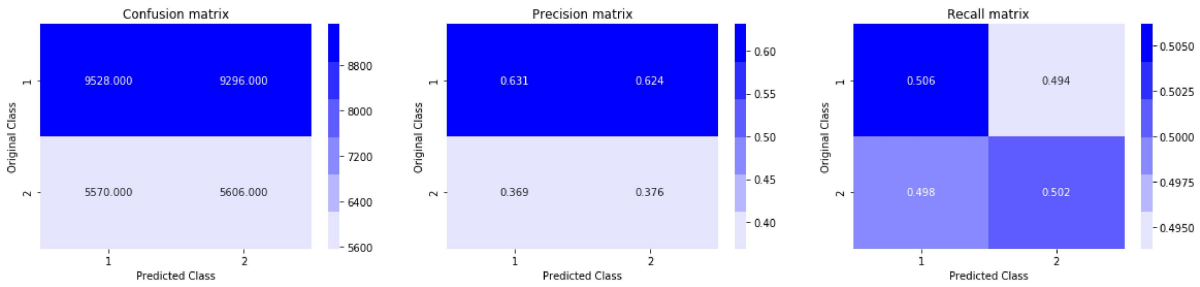
In [25]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at 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, ytick
labels=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, ytick
labels=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, ytick
```

```
labels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

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

In [26]:
```
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix

# 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 thei
r 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 0.87904178480432
```



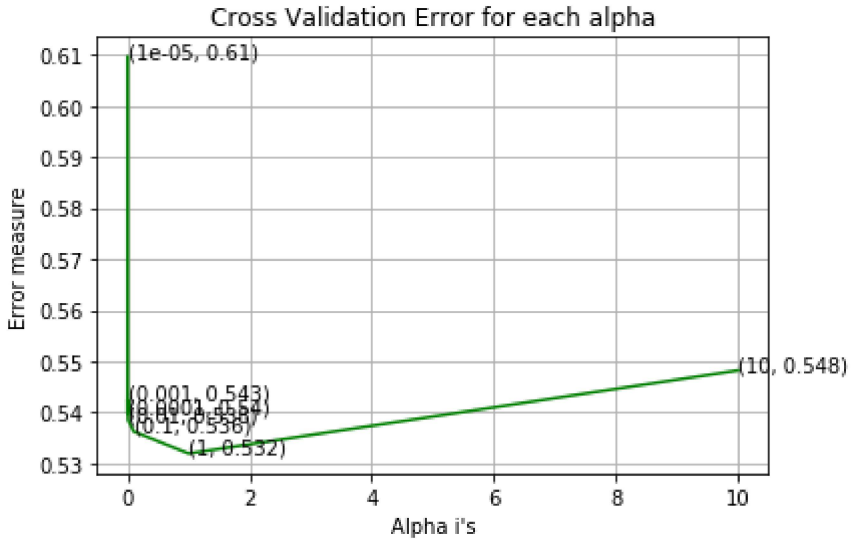
4.2 Logistic Regression with hyperparameter tuning

In [ ]:
```

```

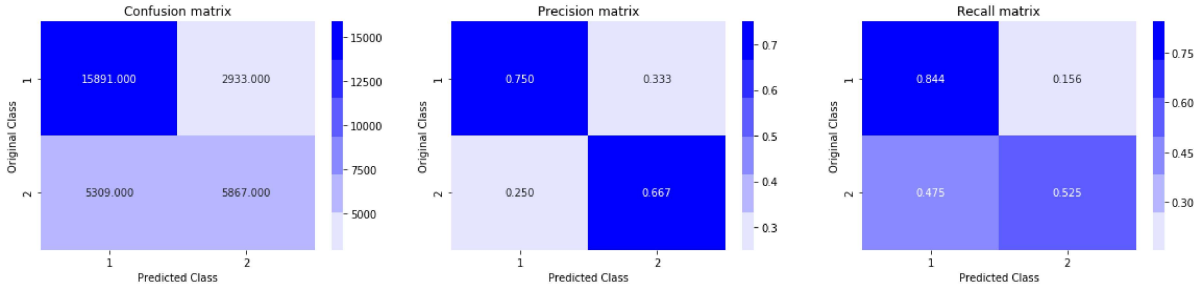
In [28]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.metrics import confusion_matrix
import seaborn as sns

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_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 i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6095979488221774
For values of alpha =  0.0001 The log loss is: 0.5397827476091186
For values of alpha =  0.001 The log loss is: 0.5425453737339507
For values of alpha =  0.01 The log loss is: 0.5384535819392331
For values of alpha =  0.1 The log loss is: 0.5362770857414854
For values of alpha =  1 The log loss is: 0.5319797468945768
For values of alpha =  10 The log loss is: 0.5482215088076026
```




```
For values of best alpha =  1 The train log loss is: 0.5240595438140132
For values of best alpha =  1 The test log loss is: 0.5319797468945768
Total number of data points : 30000
```

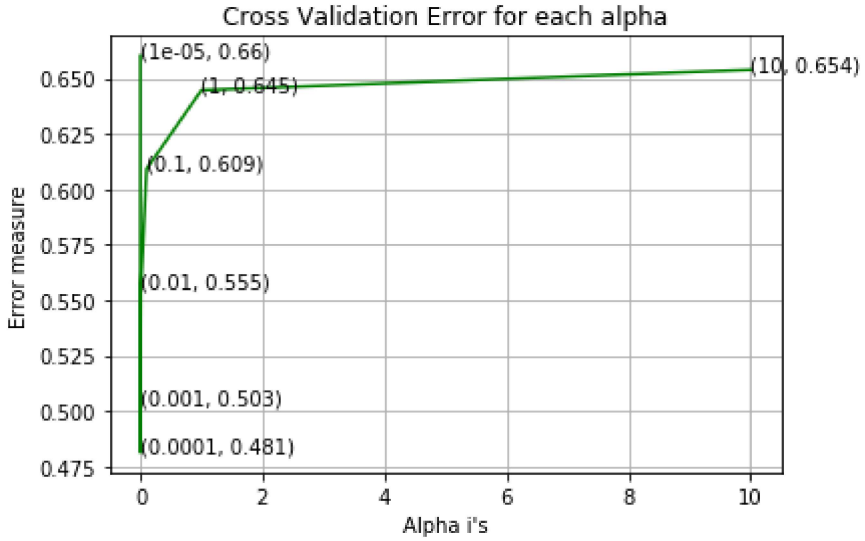


Linear SVM with hyperparameter tuning

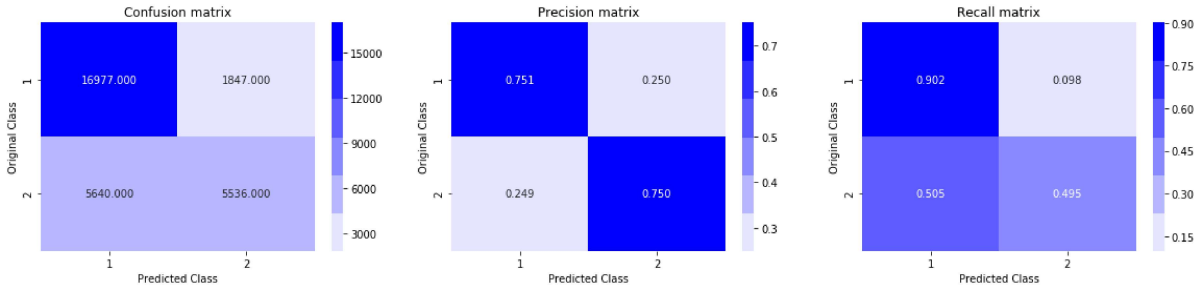
In [31]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6602902109101164
For values of alpha =  0.0001 The log loss is: 0.48115014724189514
For values of alpha =  0.001 The log loss is: 0.5027566331450263
For values of alpha =  0.01 The log loss is: 0.5553546638445868
For values of alpha =  0.1 The log loss is: 0.6091571037121764
For values of alpha =  1 The log loss is: 0.6447410602345544
For values of alpha =  10 The log loss is: 0.6539379533560806
```




```
For values of best alpha =  0.0001 The train log loss is: 0.47957830346222646
For values of best alpha =  0.0001 The test log loss is: 0.48115014724189514
Total number of data points : 30000
```



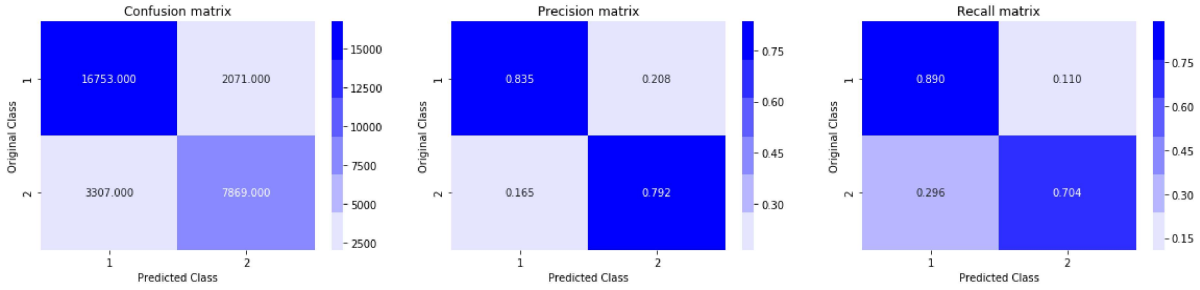
4.6 XGBoost

In [32]:
```
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, ver
bose_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:0.684939  valid-logloss:0.684997
Multiple eval metrics have been passed: 'valid-logloss' will be used for earl
y stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.615441  valid-logloss:0.616398
[20]    train-logloss:0.564624  valid-logloss:0.566199
[30]    train-logloss:0.526412  valid-logloss:0.528499
[40]    train-logloss:0.496854  valid-logloss:0.499321
[50]    train-logloss:0.473751  valid-logloss:0.476656
[60]    train-logloss:0.455603  valid-logloss:0.458891
[70]    train-logloss:0.440775  valid-logloss:0.444513
[80]    train-logloss:0.428865  valid-logloss:0.432988
[90]    train-logloss:0.419232  valid-logloss:0.42368
[100]   train-logloss:0.411041  valid-logloss:0.415784
[110]   train-logloss:0.404275  valid-logloss:0.409308
[120]   train-logloss:0.398517  valid-logloss:0.40373
[130]   train-logloss:0.393517  valid-logloss:0.398967
[140]   train-logloss:0.389406  valid-logloss:0.39506
[150]   train-logloss:0.385713  valid-logloss:0.391583
[160]   train-logloss:0.382742  valid-logloss:0.388841
[170]   train-logloss:0.380107  valid-logloss:0.386455
[180]   train-logloss:0.377615  valid-logloss:0.384169
[190]   train-logloss:0.37549   valid-logloss:0.3823
[200]   train-logloss:0.37343   valid-logloss:0.380508
[210]   train-logloss:0.371754  valid-logloss:0.379051
[220]   train-logloss:0.370022  valid-logloss:0.377568
[230]   train-logloss:0.368415  valid-logloss:0.376176
[240]   train-logloss:0.366794  valid-logloss:0.3748
[250]   train-logloss:0.365157  valid-logloss:0.373363
[260]   train-logloss:0.363428  valid-logloss:0.371849
[270]   train-logloss:0.361953  valid-logloss:0.370638
[280]   train-logloss:0.360502  valid-logloss:0.369484
[290]   train-logloss:0.359203  valid-logloss:0.368448
[300]   train-logloss:0.358062  valid-logloss:0.367578
[310]   train-logloss:0.356931  valid-logloss:0.366667
[320]   train-logloss:0.355902  valid-logloss:0.365891
[330]   train-logloss:0.354778  valid-logloss:0.365053
[340]   train-logloss:0.353698  valid-logloss:0.364199
[350]   train-logloss:0.352749  valid-logloss:0.363455
[360]   train-logloss:0.351816  valid-logloss:0.362753
[370]   train-logloss:0.350934  valid-logloss:0.362097
[380]   train-logloss:0.349962  valid-logloss:0.361392
[390]   train-logloss:0.349036  valid-logloss:0.360708
[399]   train-logloss:0.348244  valid-logloss:0.360125
The test log loss is: 0.360125698748782
```

In [33]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

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



Assignments:

XGBoost with Hyperparameter tuning

In [34]:
```
# hyperparameter tuning for max_depth and min_child_weight
min_logloss = float("Inf")
best_params = None

gridsearch_params = [
    (max_depth, min_child_weight)
    for max_depth in range(3, 10)
    for min_child_weight in range(1, 5)
]

for max_depth, min_child_weight in gridsearch_params:
    print("CV with max_depth={}, min_child_weight={}".format(
                             max_depth,
                             min_child_weight))

    # Update our parameters
    params['max_depth'] = max_depth
    params['min_child_weight'] = min_child_weight

    # Run CV
    cv_results = xgb.cv(
        params,
        d_train,
        seed=42,
        nfold=10,
        metrics={'logloss'},
        early_stopping_rounds=10
    )

    mean_logloss = cv_results['test-logloss-mean'].min()
    boost_rounds = cv_results['test-logloss-mean'].argmin()
    print("\tLogLoss {} for {} rounds".format(mean_logloss, boost_rounds))
    if mean_logloss < min_logloss:
        min_logloss = mean_logloss
        best_params = (max_depth,min_child_weight)
```

```
CV with max_depth=3, min_child_weight=1
        LogLoss 0.6274981000000001 for 9 rounds
CV with max_depth=3, min_child_weight=2
        LogLoss 0.6274981000000001 for 9 rounds
CV with max_depth=3, min_child_weight=3
        LogLoss 0.6274981000000001 for 9 rounds
CV with max_depth=3, min_child_weight=4
        LogLoss 0.6274981000000001 for 9 rounds
CV with max_depth=4, min_child_weight=1
        LogLoss 0.6218838 for 9 rounds
CV with max_depth=4, min_child_weight=2
        LogLoss 0.6218838 for 9 rounds
CV with max_depth=4, min_child_weight=3
        LogLoss 0.6218838 for 9 rounds
CV with max_depth=4, min_child_weight=4
        LogLoss 0.6218838 for 9 rounds
CV with max_depth=5, min_child_weight=1
        LogLoss 0.6184183 for 9 rounds
CV with max_depth=5, min_child_weight=2
        LogLoss 0.6184122 for 9 rounds
CV with max_depth=5, min_child_weight=3
        LogLoss 0.6184139 for 9 rounds
CV with max_depth=5, min_child_weight=4
        LogLoss 0.6184217 for 9 rounds
CV with max_depth=6, min_child_weight=1
        LogLoss 0.6147131 for 9 rounds
CV with max_depth=6, min_child_weight=2
        LogLoss 0.6147261 for 9 rounds
CV with max_depth=6, min_child_weight=3
        LogLoss 0.6147513 for 9 rounds
CV with max_depth=6, min_child_weight=4
        LogLoss 0.6147663000000001 for 9 rounds
CV with max_depth=7, min_child_weight=1
        LogLoss 0.6114481 for 9 rounds
CV with max_depth=7, min_child_weight=2
        LogLoss 0.6114756 for 9 rounds
CV with max_depth=7, min_child_weight=3
        LogLoss 0.6114808999999999 for 9 rounds
CV with max_depth=7, min_child_weight=4
        LogLoss 0.6114884 for 9 rounds
CV with max_depth=8, min_child_weight=1
        LogLoss 0.6096984 for 9 rounds
CV with max_depth=8, min_child_weight=2
        LogLoss 0.6097309000000001 for 9 rounds
CV with max_depth=8, min_child_weight=3
        LogLoss 0.609819 for 9 rounds
CV with max_depth=8, min_child_weight=4
        LogLoss 0.6098340999999999 for 9 rounds
CV with max_depth=9, min_child_weight=1
        LogLoss 0.6085344 for 9 rounds
CV with max_depth=9, min_child_weight=2
        LogLoss 0.6085900000000001 for 9 rounds
CV with max_depth=9, min_child_weight=3
        LogLoss 0.6087124 for 9 rounds
CV with max_depth=9, min_child_weight=4
        LogLoss 0.6087338 for 9 rounds
```

In [35]:
```
# best max_depth and min_child_weight
print("Best params: {}, {}".format(best_params[0], best_params[1]))
```

```
Best params: 9, 1
```

In [36]:
```
# hyperparameter tuning for subsample and colsample
gridsearch_params = [
    (subsample, colsample)
    for subsample in [i/10 for i in range(6,10)]
    for colsample in [i/10 for i in range(6,10)]
]

min_logloss = float("Inf")
best_params = None

for subsample, colsample in reversed(gridsearch_params):
    print("CV with subsample={}, colsample={}".format(
                             subsample,
                             colsample))

    # We update our parameters
    params['subsample'] = subsample
    params['colsample_bytree'] = colsample

    # Run CV
    cv_results = xgb.cv(
        params,
        d_train,
        seed=42,
        nfold=10,
        metrics={'logloss'},
        early_stopping_rounds=10
    )

    # Update best score
    mean_logloss = cv_results['test-logloss-mean'].min()
    boost_rounds = cv_results['test-logloss-mean'].argmin()
    print("\tLogLoss {} for {} rounds".format(mean_logloss, boost_rounds))
    if mean_logloss < min_logloss:
        min_logloss = mean_logloss
        best_params = (subsample,colsample)

# Best subsample and colsample
print("Best params: {}, {}".format(best_params[0], best_params[1]))
```

```
CV with subsample=0.9, colsample=0.9
        LogLoss 0.6090903 for 9 rounds
CV with subsample=0.9, colsample=0.8
        LogLoss 0.6095209 for 9 rounds
CV with subsample=0.9, colsample=0.7
        LogLoss 0.6106257 for 9 rounds
CV with subsample=0.9, colsample=0.6
        LogLoss 0.6122765 for 9 rounds
CV with subsample=0.8, colsample=0.9
        LogLoss 0.6092449999999999 for 9 rounds
CV with subsample=0.8, colsample=0.8
        LogLoss 0.6096683 for 9 rounds
CV with subsample=0.8, colsample=0.7
        LogLoss 0.6108548 for 9 rounds
CV with subsample=0.8, colsample=0.6
        LogLoss 0.6124594 for 9 rounds
CV with subsample=0.7, colsample=0.9
        LogLoss 0.6094428 for 9 rounds
CV with subsample=0.7, colsample=0.8
        LogLoss 0.6098386 for 9 rounds
CV with subsample=0.7, colsample=0.7
        LogLoss 0.6110366 for 9 rounds
CV with subsample=0.7, colsample=0.6
        LogLoss 0.6126767 for 9 rounds
CV with subsample=0.6, colsample=0.9
        LogLoss 0.6096931999999999 for 9 rounds
CV with subsample=0.6, colsample=0.8
        LogLoss 0.6101465 for 9 rounds
CV with subsample=0.6, colsample=0.7
        LogLoss 0.6112748 for 9 rounds
CV with subsample=0.6, colsample=0.6
        LogLoss 0.6129436 for 9 rounds
Best params: 0.9, 0.9
```

In [37]:
```
# hyperparameter tuning for gamma
min_logloss = float("Inf")
best_params = None

gridsearch_params=[gamma
    for gamma in [i/10.0 for i in range(0,5)]]

for gamma in gridsearch_params:
    print("CV with gamma={}".format(gamma))

    # We update our parameters
    params['gamma'] = gamma

    # Run and time CV
    cv_results = xgb.cv(
            params,
            d_train,
            seed=42,
            nfold=10,
            metrics=['logloss'],
            early_stopping_rounds=10
          )

    # Update best score
    mean_logloss = cv_results['test-logloss-mean'].min()
    boost_rounds = cv_results['test-logloss-mean'].argmin()
    print("\tLogLoss {} for {} rounds".format(mean_logloss, boost_rounds))
    if mean_logloss < min_logloss:
        min_logloss = mean_logloss
        best_params = gamma

# Best hyperparameter gamma
print("Best params: {}".format(best_params))
```

```
CV with gamma=0.0
        LogLoss 0.6129436 for 9 rounds
CV with gamma=0.1
        LogLoss 0.6129370000000001 for 9 rounds
CV with gamma=0.2
        LogLoss 0.6129262 for 9 rounds
CV with gamma=0.3
        LogLoss 0.6129252 for 9 rounds
CV with gamma=0.4
        LogLoss 0.6129197000000001 for 9 rounds
Best params: 0.4
```

In [38]:
```
# hyperparameter tuning for reg_alpha
min_logloss = float("Inf")
best_params = None


for reg_alpha in [1e-5, 1e-2, 0.1, 1, 100]:
    print("CV with reg_alpha={}".format(reg_alpha))

    # We update our parameters
    params['reg_alpha'] = reg_alpha

    # Run and time CV
    cv_results = xgb.cv(
            params,
            d_train,
            seed=42,
            nfold=10,
            metrics=['logloss'],
            early_stopping_rounds=10
          )

    # Update best score
    mean_logloss = cv_results['test-logloss-mean'].min()
    boost_rounds = cv_results['test-logloss-mean'].argmin()
    print("\tLogLoss {} for {} rounds".format(mean_logloss, boost_rounds))
    if mean_logloss < min_logloss:
        min_logloss = mean_logloss
        best_params = reg_alpha

# best reg_alpha
print("Best params: {}".format(best_params))
```

```
CV with reg_alpha=1e-05
        LogLoss 0.6129197000000001 for 9 rounds
CV with reg_alpha=0.01
        LogLoss 0.6129346 for 9 rounds
CV with reg_alpha=0.1
        LogLoss 0.6129698 for 9 rounds
CV with reg_alpha=1
        LogLoss 0.6134151999999999 for 9 rounds
CV with reg_alpha=100
        LogLoss 0.6266953 for 9 rounds
Best params: 1e-05
```

In [39]:
```
# hyperparameter tuning for number of estimators
min_logloss = float("Inf")
best_params = None

for n_estimators in [70,80, 95, 100, 140, 175, 250, 400, 600, 700, 900]:
    print("CV with n_estimators={}".format(n_estimators))

    # We update our parameters
    params['n_estimators'] = n_estimators

    # Run and time CV
    cv_results = xgb.cv(
            params,
            d_train,
            seed=42,
            nfold=10,
            metrics=['logloss'],
            early_stopping_rounds=10
          )

    # Update best score
    mean_logloss = cv_results['test-logloss-mean'].min()
    boost_rounds = cv_results['test-logloss-mean'].argmin()
    print("\tLogLoss {} for {} rounds".format(mean_logloss, boost_rounds))
    if mean_logloss < min_logloss:
        min_logloss = mean_logloss
        best_params = n_estimators
# Best n_estimators
print("Best params: {}".format(best_params))
```

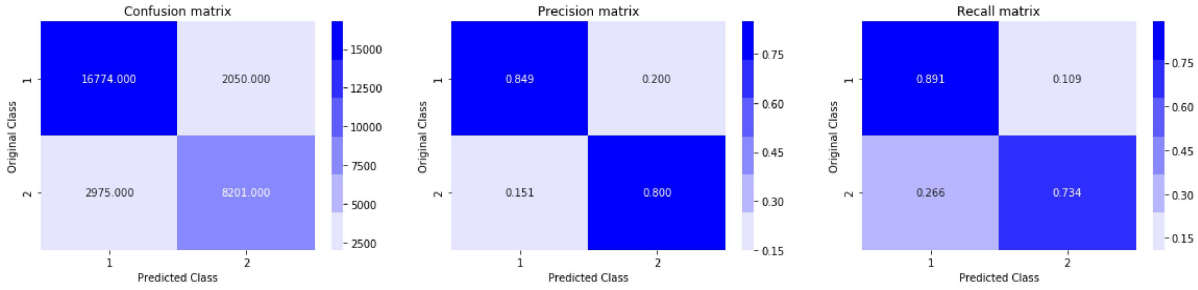
```
CV with n_estimators=70
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=80
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=95
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=100
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=140
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=175
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=250
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=400
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=600
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=700
        LogLoss 0.6266953 for 9 rounds
CV with n_estimators=900
        LogLoss 0.6266953 for 9 rounds
Best params: 70
```

In [40]:
```
# Hyperparameter tuned model with best hyperparameters found using CV and redu
ced test log-loss.

import xgboost as xgb
params= {}

params['n_estimators'] = 70
params['min_child_weight'] = 1
params['subsample'] = 0.9
params['colsample_bytree'] = 0.9
params['gamma'] = 0.4
params['reg_alpha'] = 1e-05
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.01
params['max_depth'] = 9

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, ver
bose_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:0.687866  valid-logloss:0.688163
Multiple eval metrics have been passed: 'valid-logloss' will be used for earl
y stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.640299  valid-logloss:0.643473
[20]    train-logloss:0.600396  valid-logloss:0.606252
[30]    train-logloss:0.566135  valid-logloss:0.574472
[40]    train-logloss:0.536714  valid-logloss:0.547382
[50]    train-logloss:0.511364  valid-logloss:0.524276
[60]    train-logloss:0.488844  valid-logloss:0.503956
[70]    train-logloss:0.468944  valid-logloss:0.486187
[80]    train-logloss:0.451614  valid-logloss:0.470811
[90]    train-logloss:0.436053  valid-logloss:0.457191
[100]   train-logloss:0.422159  valid-logloss:0.445168
[110]   train-logloss:0.409655  valid-logloss:0.434544
[120]   train-logloss:0.398595  valid-logloss:0.425165
[130]   train-logloss:0.388464  valid-logloss:0.416817
[140]   train-logloss:0.379359  valid-logloss:0.409405
[150]   train-logloss:0.370997  valid-logloss:0.402736
[160]   train-logloss:0.363326  valid-logloss:0.396786
[170]   train-logloss:0.356405  valid-logloss:0.39147
[180]   train-logloss:0.350105  valid-logloss:0.386645
[190]   train-logloss:0.34422   valid-logloss:0.382246
[200]   train-logloss:0.338828  valid-logloss:0.378339
[210]   train-logloss:0.333792  valid-logloss:0.374827
[220]   train-logloss:0.329264  valid-logloss:0.371653
[230]   train-logloss:0.325117  valid-logloss:0.368733
[240]   train-logloss:0.321325  valid-logloss:0.366108
[250]   train-logloss:0.317865  valid-logloss:0.363741
[260]   train-logloss:0.31462   valid-logloss:0.361642
[270]   train-logloss:0.311507  valid-logloss:0.359685
[280]   train-logloss:0.308667  valid-logloss:0.357925
[290]   train-logloss:0.305796  valid-logloss:0.356281
[300]   train-logloss:0.30321   valid-logloss:0.354791
[310]   train-logloss:0.300761  valid-logloss:0.353397
[320]   train-logloss:0.298485  valid-logloss:0.352152
[330]   train-logloss:0.296406  valid-logloss:0.351058
[340]   train-logloss:0.294248  valid-logloss:0.349963
[350]   train-logloss:0.292253  valid-logloss:0.348924
[360]   train-logloss:0.290424  valid-logloss:0.34805
[370]   train-logloss:0.288605  valid-logloss:0.347201
[380]   train-logloss:0.286852  valid-logloss:0.346474
[390]   train-logloss:0.285201  valid-logloss:0.345709
[399]   train-logloss:0.283668  valid-logloss:0.345052
The test log loss is: 0.34505281355123346
```

In [41]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

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



Steps we have performed :

1. First we did Exploratory Data Analysis on Quora Question Pair data in which we performed such as finding number of different questions, checking for duplicates, number of occurence of questions etc.
2. Then we performed some feature extraction like fuzz ratio, fuzz partial ration, longest common substring etc.
3. After performing feature extraction we applied some visualisation techniques such as pair-plot, violin plot, TSNE etc.
4. There after, we split the data set into train and test set in the ratio 70:30.
5. Then we peformed tfidf-w2vec vectorizer on pair of questions dataset and then we merged each tfidf-w2vec vectors to our advanced featured vectors.
6. In the next step we applied some machine learning algorithm such as logistic regression, support vector machines etc and found log-loss for both train and test dataset.
7. After choosing best parameters we then plotted confusion matrix, precision matrix and recall matrix for each one.
8. After that we perform Hyperparameter tuning for XGBoost and fetch the best param for each attribute.

Observations

In [49]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.title = " Model Comparision "
x.field_names = ["Model","vectorizer","Hyperparameter Tunning","Test log loss"
]
x.add_row(['Random','TFIDF w2vec','NA','0.879041'])
x.add_row(['Logistic regression','TFIDF w2vec','Done','0.53197'])
x.add_row(['Linear SVM','TFIDF w2vec','Done','0.481150'])
x.add_row(['XGBOOST','TFIDF w2vec','NA','0.360125'])
x.add_row(['XGBOOST-With Hyperparameter Tuning','TFIDF w2vec','Done','0.34505
2'])

print(x)
```

```
+-----------------------------------+------------+-----------------------+
---------------+
|               Model               |  vectorizer | Hyperparameter Tunning |
Test log loss |
+-----------------------------------+------------+-----------------------+
---------------+
|               Random              | TFIDF w2vec |           NA           |
0.879041   |
|        Logistic regression        | TFIDF w2vec |          Done          |
0.53197    |
|            Linear SVM             | TFIDF w2vec |          Done          |
0.481150   |
|              XGBOOST              | TFIDF w2vec |           NA           |
0.360125   |
| XGBOOST-With Hyperparameter Tuning | TFIDF w2vec |          Done          |
0.345052   |
+-----------------------------------+------------+-----------------------+
---------------+
```

Conclusions :

In the above analysis, we can conclude that XGBOOST performs well as compare to another models in low dimensions.