In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Quora Question Pairs

1. Business Problem

1.1 Description </h2>

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</h2>

- 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 </h2>

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 </h1>

2.1 Data </h2>

2.1.1 Data Overview </h3>

- 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 </h3>

```
"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 </h2>

2.2.1 Type of Machine Leaning Problem </h3>

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 </h3>

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 </h2>

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 [2]:

```
file_path = "/content/drive/MyDrive/Applied AI Assignments/LN Assignments/Case Study 1 Quora quest
ion pair/"

#"/content/drive/MyDrive/Applied AI Assignments/case_study_1_Quora/"
```

In [3]:

```
pip install distance
```

```
Requirement already satisfied: distance in /usr/local/lib/python3.7/dist-packages (0.1.3)
```

In [4]:

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

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

3.1 Reading data and basic stats </h2>

In [5]:

```
df = pd.read_csv(file_path +"original_train.csv")

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

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

In [6]:

```
df = df.sample(n = 10000)
```

In [7]:

```
df.to_csv(file_path+"train.csv", index=False)
```

In [8]:

```
df['id'].shape
```

Out[8]:

```
(10000,)
```

In [9]:

```
df.head()
```

Out[9]:

	id	qid1	qid2	question1	question2	is_duplicate
206552	206552	69616	310006	Where can I find free TV shows online without ...	Where can I use a credit card without the CVV ...	0
108228	108228	58375	177877	Who is going to Delhi this year for IAS prepar...	Which is the best month to go to Delhi for IAS...	0
13096	13096	25172	25173	Why is the name of Hashem (G-D) not mentioned ...	Why isn't God mentioned in the Book of Esther?	1

	id	qid1	qid2	question1	question2	is_duplicate
173278	173278	68426	30173	How do I overcome from depression without gett...	How is depression cured without a therapist?	1
174488	174488	268903	268904	How is life as a technical support engineer at...	Will Microsoft GTSC help me build my career in...	0

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 206552 to 165297
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            10000 non-null  int64 
 1   qid1          10000 non-null  int64 
 2   qid2          10000 non-null  int64 
 3   question1     10000 non-null  object
 4   question2     10000 non-null  object
 5   is_duplicate  10000 non-null  int64 
dtypes: int64(4), object(2)
memory usage: 546.9+ KB
```

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</h3>

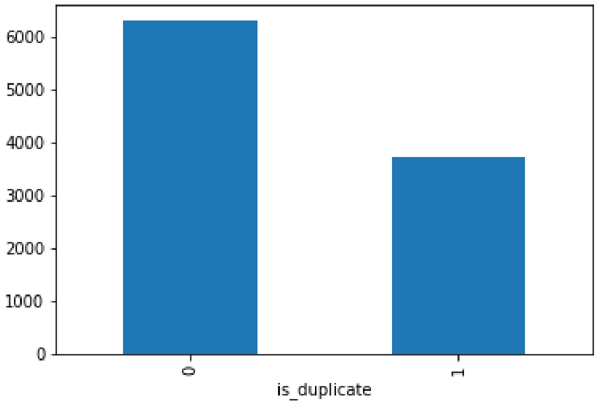
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [11]:

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

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f069682d750>
```



In [12]:

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

```
~> Total number of question pairs for training:
   10000
```

In [13]:

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

```
~> Question pairs are not Similar (is_duplicate = 0):
   62.81%

~> Question pairs are Similar (is_duplicate = 1):
   37.19%
```

3.2.2 Number of unique questions </h3>

In [14]:

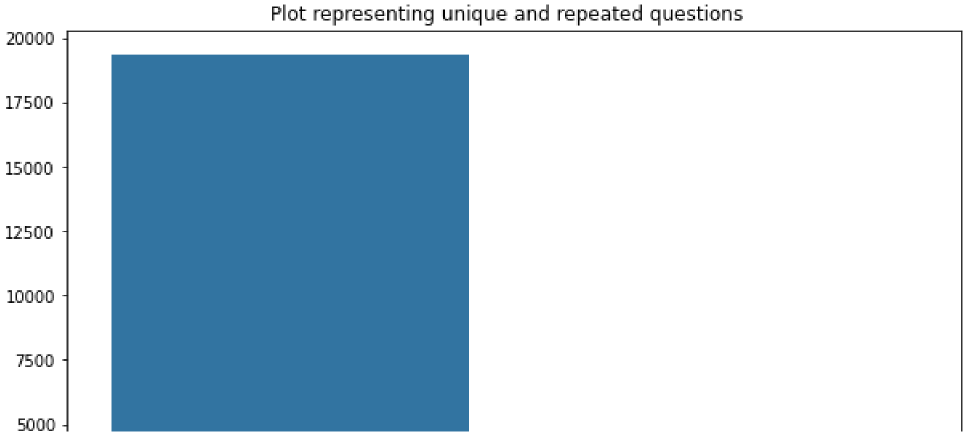
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

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

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


q_vals=qids.value_counts()

q_vals=q_vals.values
```

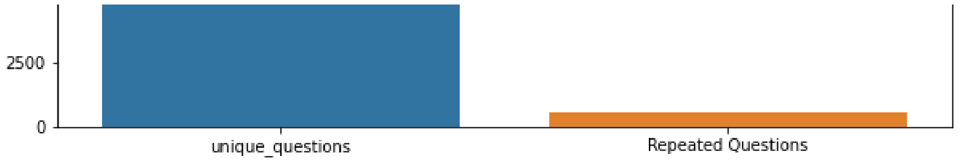
```
Total number of  Unique Questions are: 19347

Number of unique questions that appear more than one time: 575 (2.9720370083217036%)

Max number of times a single question is repeated: 5
```

In [15]:

```
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()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
```





3.2.3 Checking for Duplicates </h3>

In [16]:

```
#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])
```

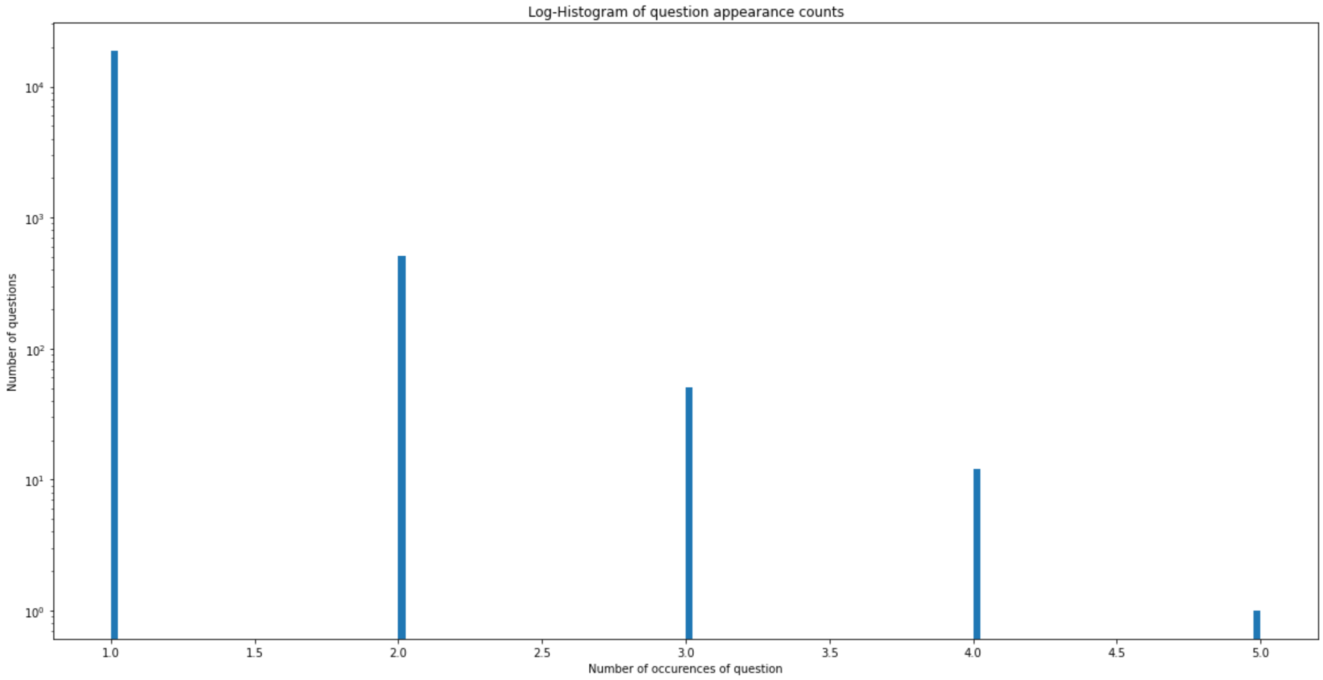
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question </h3>

In [17]:

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

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

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

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

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

plt.ylabel('Number of questions')

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

```
Maximum number of times a single question is repeated: 5
```



3.2.5 Checking for NULL values </h3>

3.2.5 Checking for NULL values </h3>

In [18]:

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

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

- There are two rows with null values in question2

In [19]:

```
# 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) </h2>

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 [20]:

```
if os.path.isfile(file_path+'df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv(file_path+"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['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(file_path+"df_fe_without_preprocessing_train.csv", index=False)

df.head()
```

Out[20]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n
206552	206552	69616	310006	Where can I find free TV shows online without ...	Where can I use a credit card without the CVV ...	0	1	1	74	54	14	11
108228	108228	58375	177877	Who is going to Delhi this year for IAS prepar...	Which is the best month to go to Delhi for IAS...	0	1	1	52	59	10	12
13096	13096	25172	25173	Why is the name of Hashem (G-D) not mentioned ...	Why isn't God mentioned in the Book of Esther?	1	1	1	72	46	14	9
173278	173278	68426	30173	How do I overcome from depression without gett...	How is depression cured without a therapist?	1	1	1	67	44	11	7
174488	174488	268903	268904	How is life as a technical support engineer at...	Will Microsoft GTSC help me build my career in...	0	1	1	62	58	11	10

In [21]:

```
df['id'].shape
```

Out[21]:

```
(10000,)
```

3.3.1 Analysis of some of the extracted features </h3>

- Here are some questions have only one single words.

In [22]:

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

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

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

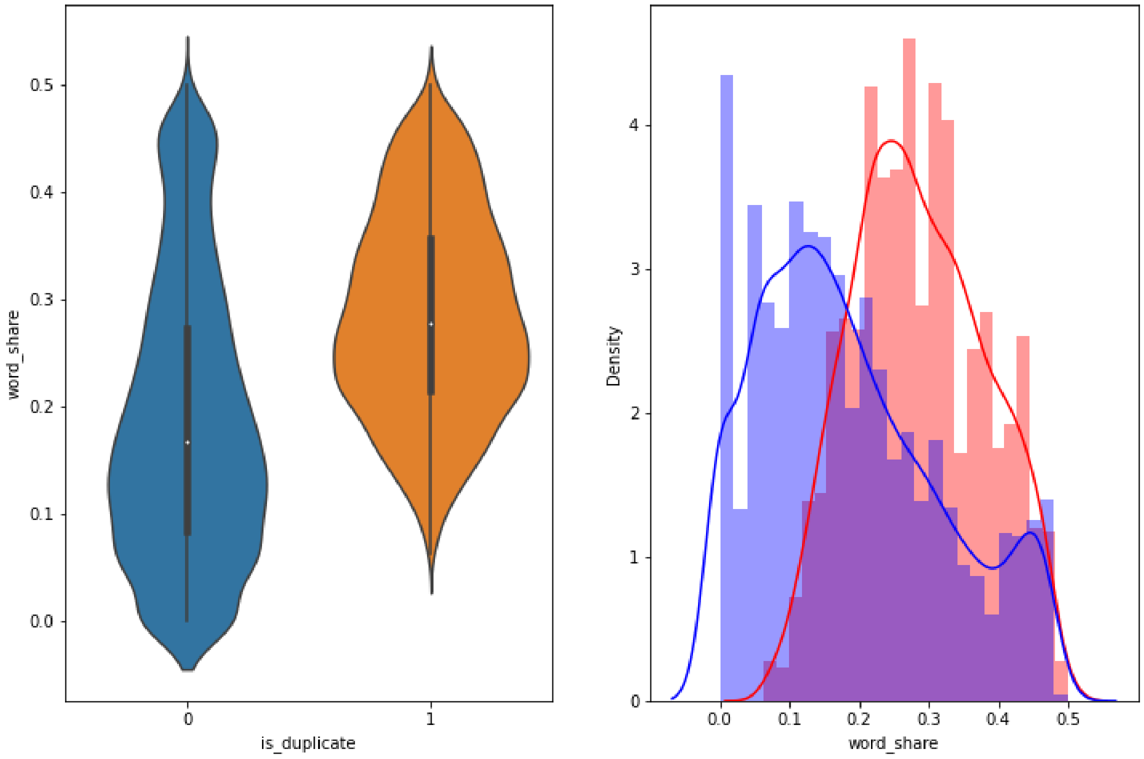
```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 3
Number of Questions with minimum length [question2] : 1
```

3.3.1.1 Feature: word_share </h4>

In [23]:

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

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

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axe
s-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axe
s-level function for histograms).
```



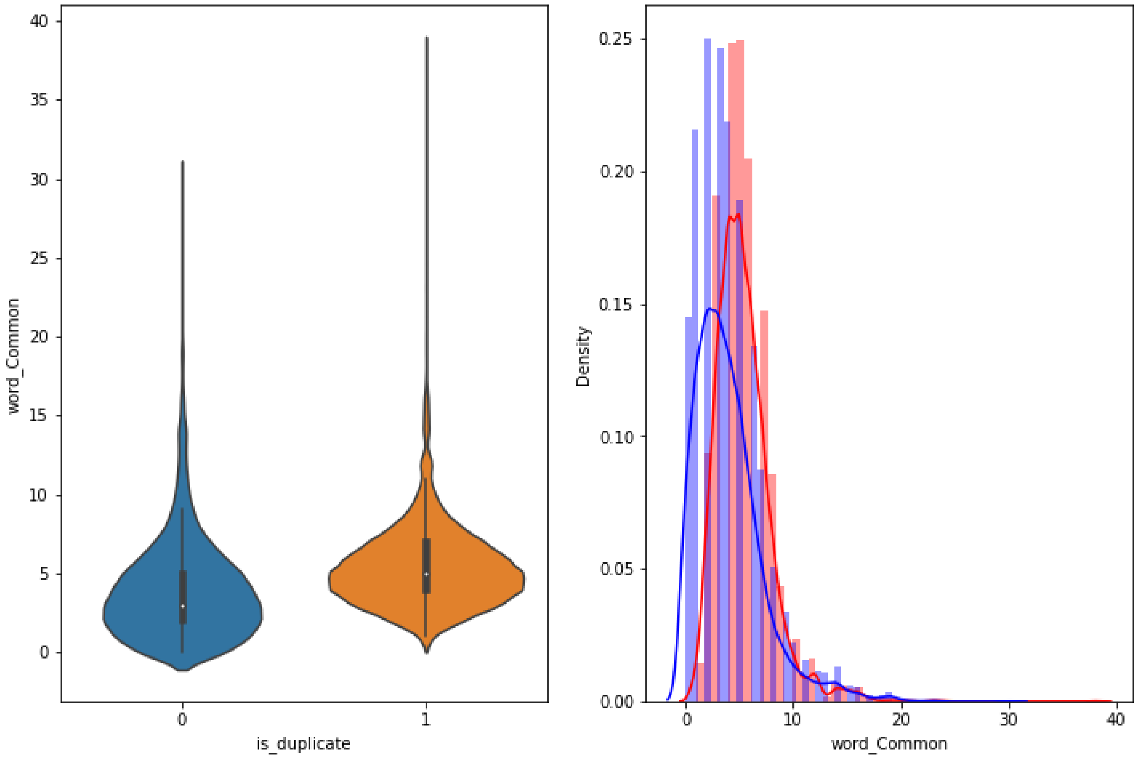
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common </h4>

In [24]:

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

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

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axe
s-level function for histograms).

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axe
s-level function for histograms).
```



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

4.EDA: Advanced Feature Extraction.

In [25]:

```
pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.7/dist-packages (0.18.0)
```

In [26]:

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
```

```
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

In [27]:

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

Out[27]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_v
206552	206552	69616	310006	Where can I find free TV shows online without ...	Where can I use a credit card without the CVV ...	0	1	1	74	54	14	11
108228	108228	58375	177877	Who is going to Delhi this year for IAS prepar...	Which is the best month to go to Delhi for IAS...	0	1	1	52	59	10	12

In [28]:

```
df['id'].shape
```

Out[28]:

```
(10000,)
```

4.1 Preprocessing of Text

- Preprocessing:
  - Removing html tags
  - Removing Punctuations
  - Performing stemming
  - Removing Stopwords

- Expanding contractions etc.

In [29]:

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[29]:

```
True
```

In [30]:

```
# 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
s")\
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").replace("'s", " own"
\
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar
)\
                           .replace("€", " euro ").replace("'ll", " will")
  x = re.sub(r"([0-9]+)000000", r"\1m", x)
  x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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

  return x
```

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

4.2 Advanced Feature Extraction (NLP and Fuzzy Features) </h2>

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 [31]:

```
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)
    # print(token_features)

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

    #Computing Fuzzy Features and Merging with Dataset

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

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

In [32]:

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

```
Extracting features for train:
token features...
fuzzy features..
```

In [33]:

```
df.columns
```

Out[33]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [34]:

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

Out[34]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_
206552	206552	69616	310006	where can i find free tv shows online without ...	where can i use a credit card without the cvv ...	0	1	1	74	54	14	11
108228	108228	58375	177877	who is going to delhi this year for ias prepar...	which is the best month to go to delhi for ias...	0	1	1	52	59	10	12

In [35]:

```
df['id'].shape
```

Out[35]:

```
(10000,)
```

4.2.1 Analysis of extracted features </h3>

4.2.1.1 Plotting Word clouds</h4>

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

In [36]:

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

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

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

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

```
Number of data points in class 1 (duplicate pairs) : 7438
Number of data points in class 0 (non duplicate pairs) : 12562
```

In [37]:

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

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))
```

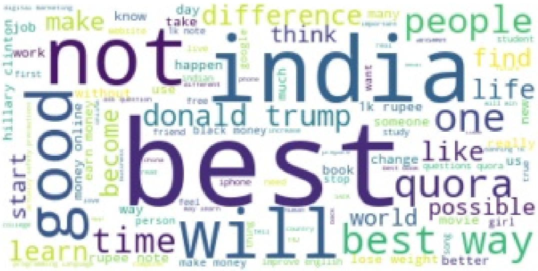
```
Total number of words in duplicate pair questions : 400291
Total number of words in non duplicate pair questions : 818818
```

Word Clouds generated from duplicate pair question's text

In [38]:

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

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



Word Clouds generated from non duplicate pair question's text

In [39]:

```
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:
```

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

In [40]:

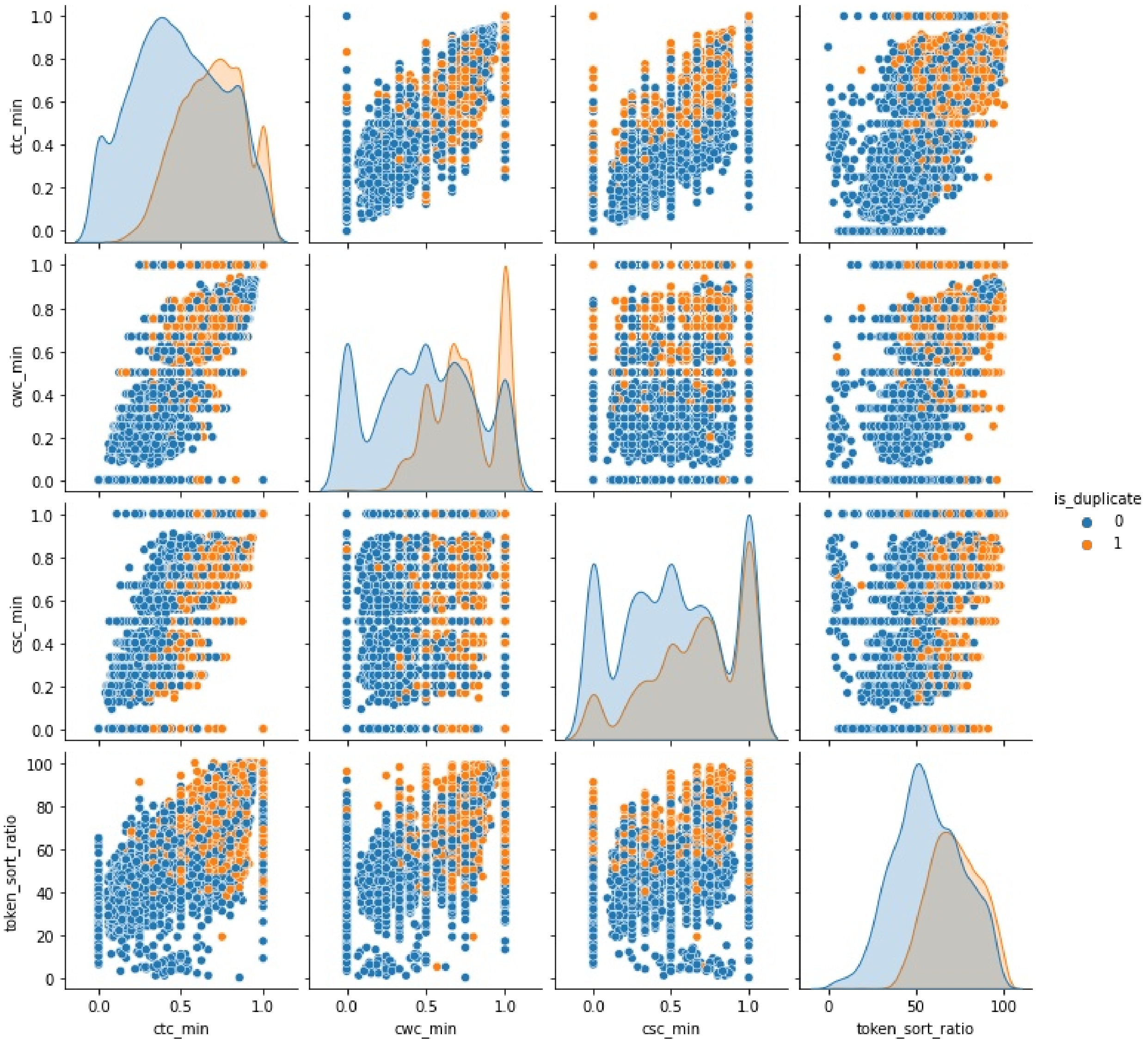
```
df.shape[0]
```

Out[40]:

```
10000
```

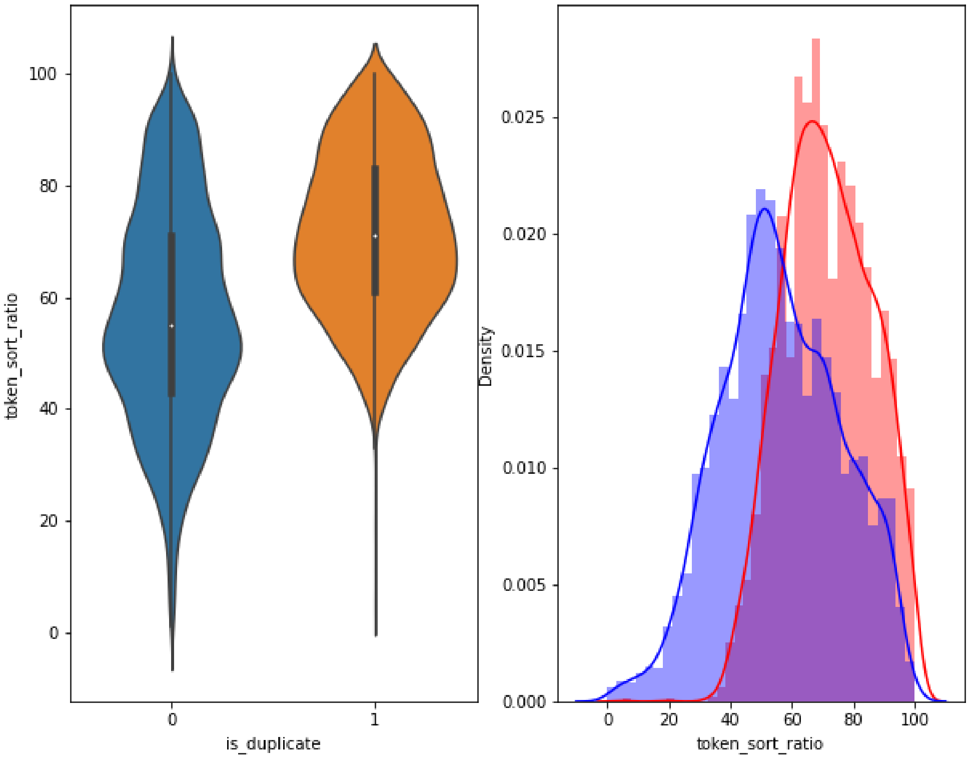
In [41]:

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



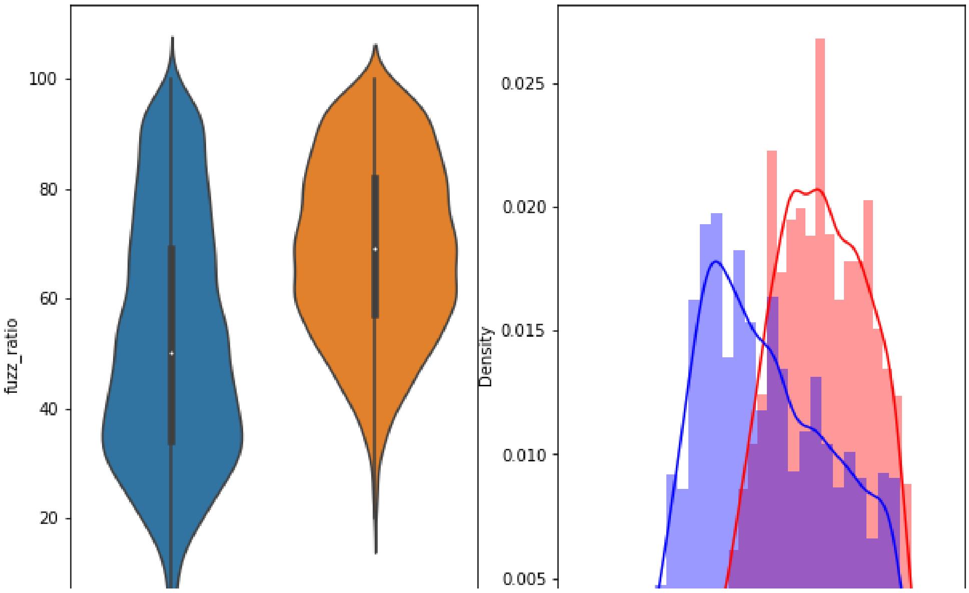
In [42]:

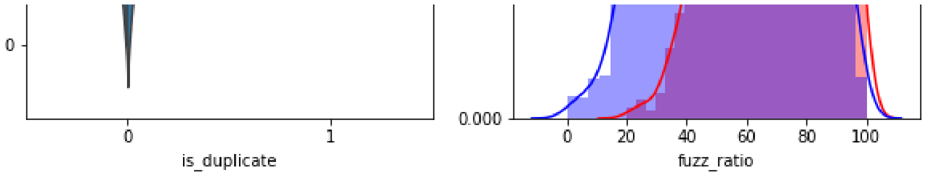
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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 [43]:

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

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

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





4.3 Visualization

In [44]:

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

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_max' ,
'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_word_eq' , 'abs_len_diff' , 'mean_len' , 'token_set_
ratio' , 'token_sort_ratio' ,  'fuzz_ratio' , 'fuzz_partial_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

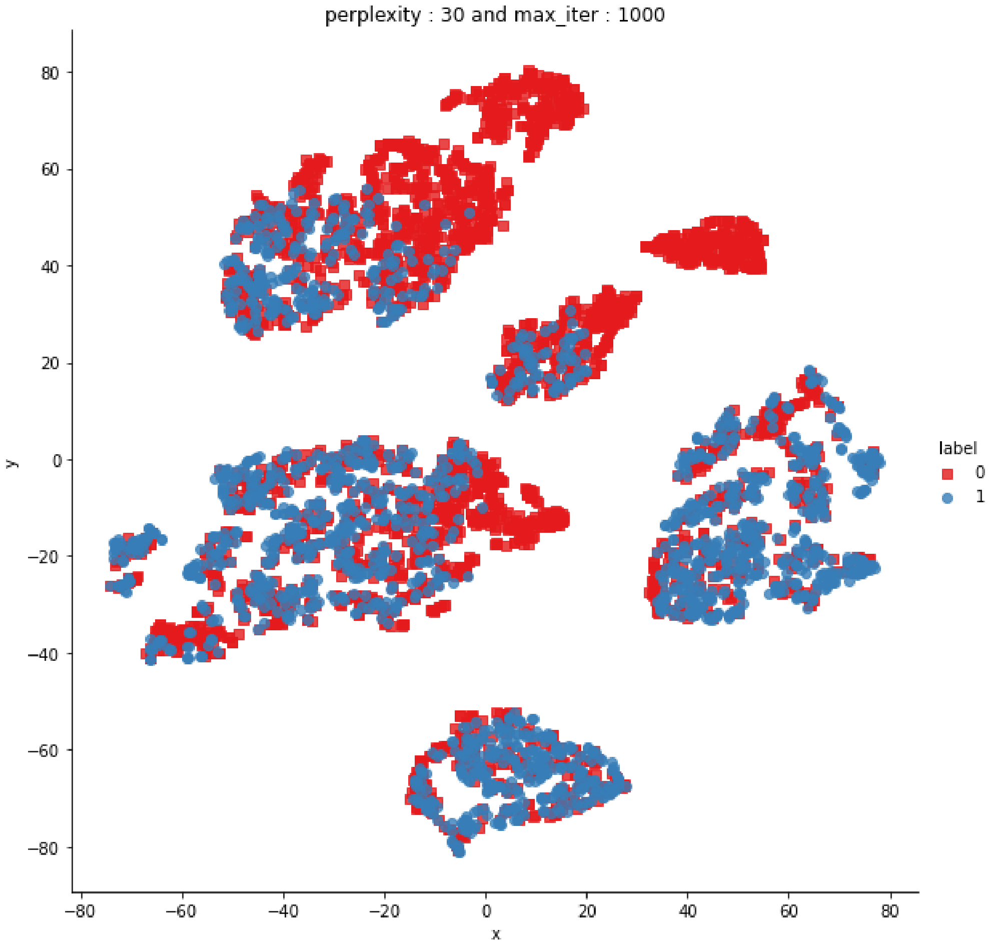
In [45]:

```
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 0.017s...
[t-SNE] Computed neighbors for 5000 samples in 0.382s...
[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: 0.132272
[t-SNE] Computed conditional probabilities in 0.333s
[t-SNE] Iteration 50: error = 83.1597748, gradient norm = 0.0495439 (50 iterations in 2.785s)
[t-SNE] Iteration 100: error = 70.5388184, gradient norm = 0.0098500 (50 iterations in 1.929s)
[t-SNE] Iteration 150: error = 68.5986633, gradient norm = 0.0053695 (50 iterations in 1.819s)
[t-SNE] Iteration 200: error = 67.7535858, gradient norm = 0.0038154 (50 iterations in 1.862s)
[t-SNE] Iteration 250: error = 67.2703552, gradient norm = 0.0038403 (50 iterations in 1.813s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.270355
[t-SNE] Iteration 300: error = 1.7787983, gradient norm = 0.0011493 (50 iterations in 1.859s)
[t-SNE] Iteration 350: error = 1.3869522, gradient norm = 0.0004775 (50 iterations in 1.863s)
[t-SNE] Iteration 400: error = 1.2232909, gradient norm = 0.0002748 (50 iterations in 1.838s)
[t-SNE] Iteration 450: error = 1.1355169, gradient norm = 0.0001849 (50 iterations in 1.833s)
[t-SNE] Iteration 500: error = 1.0818622, gradient norm = 0.0001393 (50 iterations in 1.853s)
[t-SNE] Iteration 550: error = 1.0473764, gradient norm = 0.0001151 (50 iterations in 1.875s)
[t-SNE] Iteration 600: error = 1.0237892, gradient norm = 0.0001019 (50 iterations in 1.859s)
[t-SNE] Iteration 650: error = 1.0079094, gradient norm = 0.0000901 (50 iterations in 1.832s)
[t-SNE] Iteration 700: error = 0.9969798, gradient norm = 0.0000833 (50 iterations in 1.854s)
[t-SNE] Iteration 750: error = 0.9888974, gradient norm = 0.0000865 (50 iterations in 1.858s)
[t-SNE] Iteration 800: error = 0.9828986, gradient norm = 0.0000753 (50 iterations in 1.837s)
[t-SNE] Iteration 850: error = 0.9779878, gradient norm = 0.0000710 (50 iterations in 1.890s)
[t-SNE] Iteration 900: error = 0.9736978, gradient norm = 0.0000662 (50 iterations in 1.839s)
[t-SNE] Iteration 950: error = 0.9697720, gradient norm = 0.0000648 (50 iterations in 1.867s)
[t-SNE] Iteration 1000: error = 0.9665182, gradient norm = 0.0000626 (50 iterations in 1.858s)
[t-SNE] KL divergence after 1000 iterations: 0.966518
```

In [46]:

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

# draw the plot in appropriate place in the grid
# draw the plot in appropriate place in the grid
sns.lmplot(data= df_tsne, 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()
```



5.Data splliting into Train and Test

In [47]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

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

In [48]:

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

Out[48]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_v
206552	206552	69616	310006	where can i find free tv shows online without ...	where can i use a credit card without the cvv ...	0	1	1	74	54	14	11
108228	108228	58375	177877	who is going to delhi this year for ias prepar...	which is the best month to go to delhi for ias...	0	1	1	52	59	10	12

In [49]:

```
# 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 [50]:

```
df['id'].shape
```

Out[50]:

```
(10000,)
```

In [51]:

```
y = df['is_duplicate']
X = df
```

In [52]:

```
from sklearn.model_selection import train_test_split
x_tr, x_test, y_tr, y_test = train_test_split(X, y, test_size=0.3, random_state=0, stratify = y)
```

In [53]:

```
x_tr['id'].size , x_test['id'].size
```

Out[53]:

```
(7000, 3000)
```

In [54]:

```
print('-'*20+'Training Data'+'-'*20)
print(' similar questions {}'.format(x_tr[x_tr['is_duplicate']==1]['id'].size/x_tr['id'].size))
print(' not similar questions {}'.format(x_tr[x_tr['is_duplicate']==0]
['id'].size/x_tr['id'].size))

print('-'*20+'Test Data'+'-'*20)
print(' similar questions are {}'.format(x_test[x_test['is_duplicate']==1]['id'].size/x_test['id'].
size))
print(' not similar questions are {}'.format(x_test[x_test['is_duplicate']==0]['id'].size/x_test['i
d'].size))
```

```
--------------------Training Data--------------------
 similar questions 0.37185714285714283
 not similar questions 0.6281428571428571
--------------------Test Data--------------------
 similar questions are 0.372
 not similar questions are 0.628
```

In [55]:

```
x_tr.head(2)
```

Out[55]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n
95094	95094	158674	158675	is the liberal democratic party truly liberal	what does the liberal democratic party believe...	0	1	1	46	50	7	8
220488	220488	327589	327590	what are some fun bet ideas	what are fun stakes to set when making a bet w...	0	1	1	28	63	6	13

6.Featurizing text data with tfidf weighted word-vectors

6.1 on question_1

In [56]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
que_tr_1 = list(x_tr['question1'])
que_test_1 = list(x_test['question1'])

tfidf = TfidfVectorizer(lowercase=False, max_features = 100)
tfidf.fit(que_tr_1)
```

Out[56]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=False, max_df=1.0, max_features=100,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

In [57]:

```
x_tr_que_tfidf_1 = tfidf.transform(que_tr_1)

x_test_que_tfidf_1 = tfidf.transform(que_test_1)
```

In [58]:

```
print("Shape of matrix after one hot encodig ",x_tr_que_tfidf_1.shape)
print("Shape of matrix after one hot encodig ",x_test_que_tfidf_1.shape)
```

```
Shape of matrix after one hot encodig  (7000, 100)
Shape of matrix after one hot encodig  (3000, 100)
```

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

In [59]:

```
df['id'].shape
```

Out[59]:

```
(10000,)
```

In [60]:

```
import spacy
nlp = spacy.load('en_core_web_sm') # en_vectors_web_lg, which includes over 1 million unique
vectors.
```

In [61]:

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

In [62]:

```
vecs1_tr = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(x_tr['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    n = len(doc1)
    if(n!=0):
      m = len(doc1[0].vector)
    else:
      m = 96
    mean_vec1 = np.zeros([n, m])
    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_tr.append(mean_vec1)
```

```
100%|██████████| 7000/7000 [03:56<00:00, 29.58it/s]
```

In [63]:

```
vecs1_tr[0].shape, vecs1_tr[100].shape, len(doc1), len(doc1[0].vector)
```

Out[63]:

```
((96,), (96,), 7, 96)
```

In [64]:

```
doc1[0].vector
```

Out[64]:

```
array([-1.7159698 , -0.94028306,  1.4302827 , -4.8813066 ,  5.602137  ,
        0.41269925, -2.3720655 ,  1.1801261 , -2.6673112 ,  0.47985882,
        0.0292834 , -0.5730588 , -1.4184704 ,  1.4368011 ,  4.124066  ,
       -0.11633208,  1.841193  ,  2.8167768 ,  1.6611824 ,  1.0557799 ,
       -3.7842743 ,  0.60988855, -1.2097673 , -1.5597488 , -4.5673475 ,
       -0.53190565, -4.439723  , -2.4376507 , -0.4086672 ,  1.2029594 ,
        0.15232885,  4.910077  , -1.9216218 ,  2.8664198 ,  0.74302423,
       -2.1705668 ,  0.13028345,  0.11531692,  3.9189754 ,  0.15966076,
       -1.416482  ,  1.2364417 , -0.93643475, -2.3468156 ,  4.6074243 ,
       -1.3140641 , -0.5752812 , -1.7728248 , -1.2327435 ,  0.7378238 ,
        1.7125928 ,  0.04458308, -1.7939475 , -1.401803  ,  2.467896  ,
        0.18753523,  0.07729542, -2.5627563 , -0.61417913, -0.76402867,
        1.3572025 , -0.1970613 , -1.8572826 , -0.22799663, -2.5539856 ,
        6.2849994 ,  0.28354916, -2.2318506 ,  1.4054313 , -0.16206911,
        0.72770584, -0.54049134,  1.4264905 ,  0.9644554 , -4.271908  ,
        1.7764267 ,  1.0350004 , -0.6052711 ,  2.7613888 , -0.70816153,
        0.18992624, -0.656117  , -0.67705107,  2.6094098 ,  4.0956664 ,
       -0.882318  , -2.0373385 ,  1.888547  ,  0.5793896 ,  3.584142  ,
       -3.0542512 , -2.5895586 , -0.3792294 ,  0.20076397, -0.12577087,
       -0.10814381], dtype=float32)
```

In [65]:

```
type(vecs1_tr)
```

Out[65]:

```
list
```

In [66]:

```
x_tr['tfidf_w2v_q1'] = list(vecs1_tr)
```

In [67]:

```
x_tr.head(1)
```

Out[67]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_w
95094	95094	158674	158675	is the liberal democratic party truly liberal	what does the liberal democratic party believe...	0	1	1	46	50	7	8

In [68]:

```
vecs1_test = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(x_test['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    n = len(doc1)
    if(n!=0):
      m = len(doc1[0].vector)
    else:
      m = 96
    mean_vec1 = np.zeros([n, m])
    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_test.append(mean_vec1)
```

```
100%|██████████| 3000/3000 [01:43<00:00, 29.03it/s]
```

In [69]:

```
x_test['tfidf_w2v_q1'] = list(vecs1_test)
```

In [70]:

```
vecs1_test[0].shape
```

Out[70]:

```
(96,)
```

In [71]:

```
x_test.head(1)
```

Out[71]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_v
144592	144592	74372	228794	why does human life exist on earth	is there a place on earth where no life exists	0	1	1	35	47	7	10

6.2 on question_2

In [72]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
que_tr_2 = list(x_tr['question2'])
que_test_2 = list(x_test['question2'])

tfidf = TfidfVectorizer(lowercase=False,  max_features = 100)
tfidf.fit(que_tr_2)
```

Out[72]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=False, max_df=1.0, max_features=100,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

In [73]:

```
x_tr_que_tfidf_2 = tfidf.transform(que_tr_2)

x_test_que_tfidf_2 = tfidf.transform(que_test_2)
```

In [74]:

```
print("Shape of matrix after one hot encodig ",x_tr_que_tfidf_1.shape)
print("Shape of matrix after one hot encodig ",x_test_que_tfidf_1.shape)
```

```
Shape of matrix after one hot encodig  (7000, 100)
Shape of matrix after one hot encodig  (3000, 100)
```

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

In [75]:

```
df['id'].shape
```

Out[75]:

```
(10000,)
```

In [76]:

```
import spacy
nlp = spacy.load('en_core_web_sm') # en_vectors_web_lg, which includes over 1 million unique
vectors.
```

In [77]:

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

In [78]:

```
vecs2_tr = []
for qu2 in tqdm(list(x_tr['question2'])):
    doc2 = nlp(qu2)
    n = len(doc2)
    if(n!=0):
      m = len(doc2[0].vector)
    else:
      m = 96
    mean_vec2 = np.zeros([n, m])
    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_tr.append(mean_vec2)
```

```
100%|██████████| 7000/7000 [04:04<00:00, 28.66it/s]
```

In [79]:

```
x_tr['tfidf_w2v_q2'] = list(vecs2_tr)
```

In [80]:

```
x_tr.head(1)
```

Out[80]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_w
				is the	what does							

95094	95094 **id**	158671 **qid1**	158672 **qid2**	liberal democrat **question1** party truly liberal	the liberal democratic **question2** party believe...	0 **is_duplicate**	**freq_qid1**	**freq_qid2**	46 **q1len**	52 **q2len**	7 **q1_n_words**	8 **q2_n_w**

In [81]:

```
vecs2_test = []
for qu2 in tqdm(list(x_test['question2'])):
    doc2 = nlp(qu2)
    n = len(doc2)
    if(n!=0):
      m = len(doc2[0].vector)
    else:
      m = 96
    mean_vec2 = np.zeros([n, m])
    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_test.append(mean_vec2)
```

```
100%|██████████| 3000/3000 [01:43<00:00, 29.08it/s]
```

In [82]:

```
x_test['tfidf_w2v_q2'] = list(vecs2_test)
```

In [83]:

```
x_test.head(1)
```

Out[83]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_w
144592	144592	74372	228794	why does human life exist on earth	is there a place on earth where no life exists	0	1	1	35	47	7	10

In [84]:

```
x_tr.to_csv(file_path+"x_tr.csv")
x_test.to_csv(file_path+"x_test.csv")
```

7.Creating Final Features dataframe

7.1 TFIDF-W2V

In [85]:

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

```
#     dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
# else:
#     print("download nlp_features_train.csv from drive or run previous notebook")

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

In [86]:

```
# df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
# df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
# df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
x_tr_tfidf_w2v_q1 = pd.DataFrame(x_tr.tfidf_w2v_q1.values.tolist(), index= x_tr.index)
x_tr_tfidf_w2v_q2 = pd.DataFrame(x_tr.tfidf_w2v_q2.values.tolist(), index= x_tr.index)
```

In [87]:

```
x_tr_tfidf_w2v_q1.shape, x_tr_tfidf_w2v_q2.shape
```

Out[87]:

```
((7000, 96), (7000, 96))
```

In [88]:

```
x_test_tfidf_w2v_q1 = pd.DataFrame(x_test.tfidf_w2v_q1.values.tolist(), index= x_test.index)
x_test_tfidf_w2v_q2 = pd.DataFrame(x_test.tfidf_w2v_q2.values.tolist(), index= x_test.index)
```

In [89]:

```
x_test_tfidf_w2v_q1.shape, x_test_tfidf_w2v_q2.shape
```

Out[89]:

```
((3000, 96), (3000, 96))
```

In [90]:

```
x_tr_tfidf_w2v_q1['id'] = x_tr['id']
x_tr_tfidf_w2v_q2['id'] = x_tr['id']
```

In [91]:

```
x_test_tfidf_w2v_q1['id'] = x_test['id']
x_test_tfidf_w2v_q2['id'] = x_test['id']
```

In [92]:

```
x_test_tfidf_w2v_q2.head(2)
```

Out[92]:

	0	1	2	3	4	5	6	7	8	9	
144592	32.704681	4.057020	52.485193	-29.174418	29.249085	-8.565338	21.239076	20.250900	-27.405324	44.485319	3.97
119904	-4.982040	-11.763009	6.502761	-17.936214	23.026928	-13.473953	19.332658	-3.386513	-15.488996	13.496412	-2.88

2 rows × 97 columns

In [93]:

```
x_tr.head(2)
```

Out[93]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n
95094	95094	158674	158675	is the liberal democratic party truly liberal	what does the liberal democratic party believe...	0	1	1	46	50	7	8
220488	220488	327589	327590	what are some fun bet ideas	what are fun stakes to set when making a bet w...	0	1	1	28	63	6	13

In [94]:

```
x_test.head(2)
```

Out[94]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n
144592	144592	74372	228794	why does human life exist on earth	is there a place on earth where no life exists	0	1	1	35	47	7	10
119904	119904	194577	194578	what are the worst things about bollywood	what is the worst thing about bollywood movies	1	1	1	42	47	7	8

In [95]:

```
x_tr.shape, x_test.shape
```

Out[95]:

```
((7000, 34), (3000, 34))
```

In [96]:

```
if not os.path.isfile(file_path + 'final_features_tfidf_w2v_tr.csv'):

    print(x_tr.shape)
    x_tr  = x_tr.merge(x_tr_tfidf_w2v_q1, on='id',how='left')
    x_tr  = x_tr.merge(x_tr_tfidf_w2v_q2, on='id',how='left')
    print(x_tr.shape)

    print(x_test.shape)
    x_test  = x_test.merge(x_test_tfidf_w2v_q1, on='id',how='left')
    x_test  = x_test.merge(x_test_tfidf_w2v_q2, on='id',how='left')
    print(x_test.shape)


    x_tr.to_csv(file_path + 'final_features_tfidf_w2v_tr.csv')
    x_test.to_csv(file_path + 'final_features_tfidf_w2v_test.csv')
```

```
(7000, 34)
```

```
(7000, 226)
(3000, 34)
(3000, 226)
```

In [97]:

```
x_tr.head(2)
```

Out[97]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_word
0	95094	158674	158675	is the liberal democratic party truly liberal	what does the liberal democratic party believe...	0	1	1	46	50	7	8
1	220488	327589	327590	what are some fun bet ideas	what are fun stakes to set when making a bet w...	0	1	1	28	63	6	13

2 rows × 226 columns

In [98]:

```
x_test.head(2)
```

Out[98]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_word
0	144592	74372	228794	why does human life exist on earth	is there a place on earth where no life exists	0	1	1	35	47	7	10
1	119904	194577	194578	what are the worst things about bollywood	what is the worst thing about bollywood movies	1	1	1	42	47	7	8

2 rows × 226 columns

7.2 TFIDF

In [99]:

```
x_tr.columns.values
```

Out[99]:

```
array(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share',
       'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
       'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
       'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
       'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio',
       'tfidf_w2v_q1', 'tfidf_w2v_q2', '0_x', '1_x', '2_x', '3_x', '4_x',
       '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x',
       '14_x', '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x',
       '22_x', '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x',
       '30_x', '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x',
       '38_x', '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x',
       '46_x', '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x',
       '54_x', '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x',
       '62_x', '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x',
       '70_x', '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x',
       '78_x', '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x',
       '86_x', '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x',
       '94_x', '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y',
       '7_y', '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y',
       '15_y', '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y',
       '23_y', '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y',
       '31_y', '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y',
       '39_y', '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y',
       '47_y', '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y',
       '55_y', '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y',
       '63_y', '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y',
       '71_y', '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y',
       '79_y', '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y',
       '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
       '95_y'], dtype=object)
```

In [100]:

```
x_tr = x_tr[['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share',
       'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
       'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
       'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
       'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio']]
```

In [101]:

```
x_test = x_test[['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share',
       'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
       'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
       'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
       'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio']]
```

In [102]:

```
x_tr.head(2)
```

Out[102]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_word
0	95094	158674	158675	is the liberal democratic party truly liberal	what does the liberal democratic party believe...	0	1	1	46	50	7	8
1	220488	327589	327590	what are some fun bet ideas	what are fun stakes to set when making a bet w...	0	1	1	28	63	6	13

In [103]:

```
x_tr_que_tfidf_1.shape, x_tr_que_tfidf_2.shape, x_test_que_tfidf_1.shape, x_test_que_tfidf_2.shape
```

Out[103]:

```
((7000, 100), (7000, 100), (3000, 100), (3000, 100))
```

In [104]:

```
# df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
# df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
# df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
x_tr_que_tfidf_1 = pd.DataFrame(x_tr_que_tfidf_1.todense())
x_tr_que_tfidf_2 = pd.DataFrame(x_tr_que_tfidf_2.todense())

x_test_que_tfidf_1 = pd.DataFrame(x_test_que_tfidf_1.todense())
x_test_que_tfidf_2 = pd.DataFrame(x_test_que_tfidf_2.todense())
```

In [105]:

```
x_tr_que_tfidf_1.shape, x_tr_que_tfidf_2.shape, x_test_que_tfidf_1.shape, x_test_que_tfidf_2.shape
```

Out[105]:

```
((7000, 100), (7000, 100), (3000, 100), (3000, 100))
```

In [106]:

```
x_tr_que_tfidf_1['id'] = x_tr['id']
x_tr_que_tfidf_2['id'] = x_tr['id']
```

In [107]:

```
x_test_que_tfidf_1['id'] = x_test['id']
x_test_que_tfidf_2['id'] = x_test['id']
```

In [108]:

```
x_tr_que_tfidf_1.head(2)
```

Out[108]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.517875	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2 rows × 101 columns

In [109]:

```
x_test_que_tfidf_2.head(2)
```

Out[109]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.809968	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2 rows × 101 columns

In [110]:

```
x_tr.shape, x_test.shape
```

Out[110]:

```
((7000, 32), (3000, 32))
```

In [111]:

```
if not os.path.isfile(file_path + 'final_features_tfidf_tr.csv'):

    print(x_tr.shape)
    x_tr  = x_tr.merge(x_tr_que_tfidf_1, on='id',how='left')
    x_tr  = x_tr.merge(x_tr_que_tfidf_2, on='id',how='left')
    print(x_tr.shape)

    print(x_test.shape)
    x_test  = x_test.merge(x_test_que_tfidf_1, on='id',how='left')
    x_test  = x_test.merge(x_test_que_tfidf_2, on='id',how='left')
    print(x_test.shape)


    x_tr.to_csv(file_path + 'final_features_tfidf_tr.csv')
    x_test.to_csv(file_path + 'final_features_tfidf_test.csv')
```

```
(7000, 32)
(7000, 232)
(3000, 32)
(3000, 232)
```

In [112]:

```
x_tr.head(2)
```

Out[112]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_word
0	95094	158674	158675	is the liberal democratic party truly liberal	what does the liberal democratic party believe...	0	1	1	46	50	7	8
1	220488	327589	327590	what are some fun bet ideas	what are fun stakes to set when making a bet w...	0	1	1	28	63	6	13

2 rows × 232 columns

In [113]:

```
x_tr.isna().any().values
```

Out[113]:

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False])
```

In [114]:

```
x_test.head(2)
```

Out[114]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_word
0	144592	74372	228794	why does human life exist on earth	is there a place on earth where no life exists	0	1	1	35	47	7	10
1	119904	194577	194578	what are the worst things about bollywood	what is the worst thing about bollywood movies	1	1	1	42	47	7	8

2 rows × 232 columns

8.Saving to DB and Loading from DB: Reading data from file and storing into sql table

In [115]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve

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

8.1 TFIDF - W2V

8.1.1 Train data

In [116]:

```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None


def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the databse:")
    tables = table_names.fetchall()
    print(tables[0][0])
    print(tables)
    return(len(tables))
```

In [117]:

```
#Creating db file from csv
import sqlalchemy
if not os.path.isfile(file_path + 'tfidf_w2v_tr.db'):
  print("not present in drive")
  disk_engine = sqlalchemy.create_engine('sqlite:///'+file_path+'tfidf_w2v_tr.db')
  start = dt.datetime.now()
  chunksize = 180000
  j = 0
  index_start = 1
  for df in pd.read_csv(file_path+'final_features_tfidf_w2v_tr.csv',
                        names=['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
                               'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
                               'q2_n_words', 'word_Common', 'word_Total', 'word_share',
                               'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
                               'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
                               'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
                               'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio',
                               'tfidf_w2v_q1', 'tfidf_w2v_q2', '0_x', '1_x', '2_x', '3_x', '4_x',
                               '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x',
                               '14_x', '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x',
                               '22_x', '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x',
                               '30_x', '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x',
                               '38_x', '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x',
                               '46_x', '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x',
                               '54_x', '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x',
                               '62_x', '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x',
                               '70_x', '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x',
                               '78_x', '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x',
                               '86_x', '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x',
                               '94_x', '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y',
                               '7_y', '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y',
                               '15_y', '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y',
                               '23_y', '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y',
                               '31_y', '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y',
                               '39_y', '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y',
                               '47_y', '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y',
                               '55_y', '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y',
                               '63_y', '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y',
                               '71_y', '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y',
                               '79_y', '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y',
                               '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
                               '95_y'],
                        chunksize=chunksize, iterator=True, encoding='utf-8', ):
      df.index += index_start
      j+=1
      print('{} rows'.format(j*chunksize))
      df.to_sql('data', disk_engine, if_exists='append')
      index_start = df.index[-1] + 1
```

```
not present in drive
180000 rows
```

In [118]:

```
read_db = file_path + 'tfidf_w2v_tr.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
data
[('data',)]
```

In [119]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)
        data =pd.read_sql_query("""SELECT * From data ORDER BY RANDOM() LIMIT 100001;""", conn_r)

        # for selecting random points
#         data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn_r)
        conn_r.commit()
        conn_r.close()
```

In [120]:

```
data.shape
```

Out[120]:

```
(7001, 227)
```

In [121]:

```
data[data['is_duplicate']=='is_duplicate']
```

Out[121]:

	index	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_word
1094	NaN	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words

1 rows × 227 columns

In [122]:

```
data = data.drop(data.index[data['is_duplicate']=='is_duplicate'])
```

In [123]:

```
data.head(2)
```

Out[123]:

	index	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_
0	6319.0	98977	86348	31174	what are some good lyric prank songs for your ...	what is good song for a best friend lyric prank	1	1	1	58	47	11	9
1	4854.0	216800	3066	26378	how can i improve my writing skills for writin...	how do i improve my english writing and speaki...	1	1	1	55	56	11	10

2 rows × 227 columns

In [124]:

```
data[pd.isnull(data).any(axis=1)]
```

Out[124]:

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

0 rows × 227 columns

In [125]:

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

In [126]:

```
data.shape, y_train.shape
data.shape, y_train.shape
```

Out[126]:

```
((7000, 224), (7000,))
```

In [127]:

```
data.head(2)
```

Out[127]:

	qid1	qid2	question1	question2	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_
0	86348	31174	what are some good lyric prank songs for your ...	what is good song for a best friend lyric prank	1	1	58	47	11	9	4.0	20.0
1	3066	26378	how can i improve my writing skills for writin...	how do i improve my english writing and speaki...	1	1	55	56	11	10	5.0	20.0

2 rows × 224 columns

In [128]:

```
data.columns.values
```

Out[128]:

```
array(['qid1', 'qid2', 'question1', 'question2', 'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'tfidf_w2v_q1',
       'tfidf_w2v_q2', '0_x', '1_x', '2_x', '3_x', '4_x', '5_x', '6_x',
       '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x', '14_x',
       '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x', '22_x',
       '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x', '30_x',
       '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x', '38_x',
       '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x', '46_x',
       '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x', '54_x',
       '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x', '62_x',
       '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x', '70_x',
       '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x', '78_x',
       '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x', '86_x',
       '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x', '94_x',
       '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y',
       '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y',
       '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y',
       '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y',
       '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y',
       '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y',
       '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y',
       '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y',
       '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y',
       '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y',
       '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y',
       '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y'],
      dtype=object)
```

In [129]:

```
train_data_tfidf_w2v_tr = data[[  'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio',
        '0_x', '1_x', '2_x', '3_x', '4_x', '5_x', '6_x',
       '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x', '14_x',
       '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x', '22_x',
       '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x', '30_x',
       '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x', '38_x',
       '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x', '46_x',
       '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x', '54_x',
       '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x', '62_x',
       '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x', '70_x',
       '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x', '78_x',
       '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x', '86_x',
       '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x', '94_x',
       '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y',
       '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y',
       '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y',
       '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y',
       '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y',
       '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y',
       '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y',
       '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y',
       '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y',
       '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y',
       '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y',
       '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y']]
```

In [130]:

```
train_data_tfidf_w2v_tr.head(2)
```

Out[130]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	freq
0	1	1	58	47	11	9	4.0	20.0	0.2	2	0
1	1	1	55	56	11	10	5.0	20.0	0.25	2	0

2 rows × 218 columns

In [131]:

```
train_data_tfidf_w2v_tr.shape, y_train.shape
```

Out[131]:

```
((7000, 218), (7000,))
```

8.1.2 Test data

In [132]:

```
#Creating db file from csv
import sqlalchemy
if not os.path.isfile(file_path + 'tfidf_w2v_test.db'):
  print("not present in drive")
  disk_engine = sqlalchemy.create_engine('sqlite:///'+file_path+'tfidf_w2v_test.db')
  start = dt.datetime.now()
  chunksize = 180000
  j = 0
  index_start = 1
  for df in pd.read_csv(file_path+'final_features_tfidf_w2v_test.csv',
                        names=['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
                              'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
                              'q2_n_words', 'word_Common', 'word_Total', 'word_share',
                              'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
                              'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
                              'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
                              'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio',
                              'tfidf_w2v_q1', 'tfidf_w2v_q2', '0_x', '1_x', '2_x', '3_x', '4_x',
                              '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x',
                              '14_x', '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x',
                              '22_x', '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x',
                              '30_x', '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x',
                              '38_x', '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x',
                              '46_x', '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x',
                              '54_x', '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x',
                              '62_x', '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x',
                              '70_x', '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x',
                              '78_x', '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x',
                              '86_x', '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x',
                              '94_x', '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y',
                              '7_y', '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y',
                              '15_y', '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y',
                              '23_y', '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y',
                              '31_y', '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y',
                              '39_y', '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y',
                              '47_y', '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y',
                              '55_y', '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y',
                              '63_y', '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y',
                              '71_y', '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y',
                              '79_y', '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y',
                              '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
                              '95_y'],
                        chunksize=chunksize, iterator=True, encoding='utf-8', ):
      df.index += index_start
      j+=1
      print('{} rows'.format(j*chunksize))
      df.to_sql('data', disk_engine, if_exists='append')
      index_start = df.index[-1] + 1
```

```
not present in drive
180000 rows
```

In [133]:

```
read_db = file_path + 'tfidf_w2v_test.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
data
[('data',)]
```

In [134]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)
        data =pd.read_sql_query("""SELECT * From data ORDER BY RANDOM() LIMIT 100001;""", conn_r)

        # for selecting random points
#         data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn_r)
        conn_r.commit()
        conn_r.close()
```

In [135]:

```
data.shape
```

Out[135]:

```
(3001, 227)
```

In [136]:

```
data[data['is_duplicate']=='is_duplicate']
```

Out[136]:

	index	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
1779	NaN	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words

1 rows × 227 columns

In [137]:

```
data = data.drop(data.index[data['is_duplicate']=='is_duplicate'])
```

In [138]:

```
data.shape
```

Out[138]:

```
(3000, 227)
```

In [139]:

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

In [140]:

```
data.shape, y_test.shape
```

Out[140]:

```
((3000, 224), (3000,))
```

In [141]:

```
data.head(2)
```

Out[141]:

	qid1	qid2	question1	question2	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wo
0	480698	480699	should a skinny guy do cycling and exercise w...	do skinny people get drunk more easily	1	1	87	39	18	7	4.0	22.
1	107697	107698	how did aristotle own and galileo	how do the beliefs of the atomic	1	1	78	77	12	13	4.0	24.

	qid1	qid2	own question1 theories...	theory question2 from d...	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wo

2 rows × 224 columns

In [142]:

```
data.columns.values
```

Out[142]:

```
array(['qid1', 'qid2', 'question1', 'question2', 'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'tfidf_w2v_q1',
       'tfidf_w2v_q2', '0_x', '1_x', '2_x', '3_x', '4_x', '5_x', '6_x',
       '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x', '14_x',
       '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x', '22_x',
       '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x', '30_x',
       '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x', '38_x',
       '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x', '46_x',
       '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x', '54_x',
       '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x', '62_x',
       '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x', '70_x',
       '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x', '78_x',
       '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x', '86_x',
       '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x', '94_x',
       '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y',
       '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y',
       '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y',
       '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y',
       '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y',
       '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y',
       '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y',
       '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y',
       '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y',
       '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y',
       '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y',
       '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y'],
      dtype=object)
```

In [143]:

```
test_data_tfidf_w2v_test = data[[  'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio',
        '0_x', '1_x', '2_x', '3_x', '4_x', '5_x', '6_x',
       '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x', '14_x',
       '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x', '22_x',
       '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x', '30_x',
       '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x', '38_x',
       '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x', '46_x',
       '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x', '54_x',
       '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x', '62_x',
       '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x', '70_x',
       '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x', '78_x',
       '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x', '86_x',
       '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x', '94_x',
       '95_x', '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y',
       '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y',
       '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y',
       '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y',
       '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y',
       '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y',
       '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y',
       '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y',
       '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y',
       '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y',
       '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y',
       '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y']]
```

In [144]:

```
test_data_tfidf_w2v_test.head(2)
```

Out[144]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q
0	1	1	87	39	18	7	4.0	22.0	0.18181818181818182	2
1	1	1	78	77	12	13	4.0	24.0	0.16666666666666666	2

2 rows × 218 columns

In [145]:

```
test_data_tfidf_w2v_test.shape, y_test.shape
```

Out[145]:

```
((3000, 218), (3000,))
```

8.1.3 Converting String to Numerics

In [146]:

```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
# https://stackoverflow.com/questions/40790031/pandas-to-numeric-find-out-which-string-it-was-unab
le-to-parse
cols = list(train_data_tfidf_w2v_tr.columns)
for i in cols:
    train_data_tfidf_w2v_tr[i] = train_data_tfidf_w2v_tr[i].apply(pd.to_numeric, errors='coerce')
    print(i)
```

```
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
0_x
1_x
```

```
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
10_x
11_x
12_x
13_x
14_x
15_x
16_x
17_x
18_x
19_x
20_x
21_x
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
39_x
40_x
41_x
42_x
43_x
44_x
45_x
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
65_x
66_x
67_x
68_x
69_x
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
```

```
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
22_y
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
40_y
41_y
42_y
43_y
44_y
45_y
46_y
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
```

```
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
```

In [147]:

```
y_train[:10]
```

Out[147]:

```
array(['1', '1', '1', '0', '0', '0', '1', '0', '1', '0'], dtype=object)
```

In [148]:

```
y_train = pd.Series(map(int,list(y_train)))
```

In [149]:

```
cols = list(test_data_tfidf_w2v_test.columns)
for i in cols:
    test_data_tfidf_w2v_test[i] = test_data_tfidf_w2v_test[i].apply(pd.to_numeric , errors='coerce'
)
    print(i)
```

```
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
cwc_min
cwc_max
csc_min
csc_max
ctc_min
```

```
ctc_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
0_x
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
10_x
11_x
12_x
13_x
14_x
15_x
16_x
17_x
18_x
19_x
20_x
21_x
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
39_x
40_x
41_x
42_x
43_x
44_x
45_x
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
65_x
66_x
```

```
66_x
67_x
68_x
69_x
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
22_y
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
40_y
41_y
42_y
43_y
44_y
45_y
46_y
47_y
```

```
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
```

In [150]:

```
y_test[1:10]
```

Out[150]:

```
array(['0', '0', '0', '0', '0', '1', '1', '0', '0'], dtype=object)
```

In [151]:

```
y_test = pd.Series(map(int,list(y_test)))
```

In [152]:

```
x_tr_set1 = train_data_tfidf_w2v_tr
x_test_set1 = test_data_tfidf_w2v_test
y_tr_set1 = y_train
y_test_set1 = y_test
```

In [153]:

```
x_tr_set1.shape, x_test_set1.shape, y_tr_set1.shape, y_test_set1.shape
```

Out[153]:

```
((7000, 218), (3000, 218), (7000,), (3000,))
```

8.2 TFIDF

8.2.1 Train data

In [154]:

```
#Creating db file from csv
import sqlalchemy
if not os.path.isfile(file_path + 'tfidf_tr.db'):
  print("not present in drive")
  disk_engine = sqlalchemy.create_engine('sqlite:///'+file_path+'tfidf_tr.db')
  start = dt.datetime.now()
  chunksize = 180000
  j = 0
  index_start = 1
  for df in pd.read_csv(file_path+'final_features_tfidf_tr.csv',
                        names=['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
                               'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
                               'q2_n_words', 'word_Common', 'word_Total', 'word_share',
                               'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
                               'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
                               'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
                               'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio',
                               '0_x', '1_x', '2_x', '3_x', '4_x',
                               '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x',
                               '14_x', '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x',
                               '22_x', '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x',
                               '30_x', '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x',
                               '38_x', '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x',
                               '46_x', '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x',
                               '54_x', '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x',
                               '62_x', '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x',
                               '70_x', '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x',
                               '78_x', '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x',
                               '86_x', '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x',
                               '94_x', '95_x', '96_x', '97_x', '98_x', '99_x',
                                '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y',
                               '7_y', '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y',
                               '15_y', '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y',
                               '23_y', '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y',
                               '31_y', '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y',
                               '39_y', '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y',
                               '47_y', '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y',
                               '55_y', '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y',
                               '63_y', '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y',
                               '71_y', '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y',
                               '79_y', '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y',
                               '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
                               '95_y', '96_y', '97_y', '98_y', '99_y'],
                        chunksize=chunksize, iterator=True, encoding='utf-8', ):
      df.index += index_start
      j+=1
      print('{} rows'.format(j*chunksize))
      df.to_sql('data', disk_engine, if_exists='append')
      index_start = df.index[-1] + 1
```

```
not present in drive
180000 rows
```

In [155]:

```
read_db = file_path + 'tfidf_tr.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
```

```
data
[('data',)]
```

In [156]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)
        data =pd.read_sql_query("""SELECT * From data ORDER BY RANDOM() LIMIT 100001;""", conn_r)

        # for selecting random points
#         data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn_r)
        conn_r.commit()
        conn_r.close()
```

In [157]:

```
data.shape
```

Out[157]:

```
(7001, 233)
```

In [158]:

```
data[data['is_duplicate']=='is_duplicate']
```

Out[158]:

	index	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
5848	NaN	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words

1 rows × 233 columns

In [159]:

```
data.head(2)
```

Out[159]:

	index	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n
0	711.0	244984	290121	9053	what are your favorite quotes or sayings	what are some of your favorite quotes	1	1	1	41	38	7	7
1	6942.0	230421	213704	32541	what is the difference between infatuation l...	how do you discern between infatuation and love	1	1	1	50	48	8	8

2 rows × 233 columns

In [160]:

```
data = data.drop(data.index[data['is_duplicate']=='is_duplicate'])
```

In [161]:

```
data.shape
```

Out[161]:

```
(7000, 233)
```

In [162]:

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

In [163]:

```
data.shape, y_train.shape
```

Out[163]:

```
((7000, 230), (7000,))
```

In [164]:

```
data.head(2)
```

Out[164]:

	qid1	qid2	question1	question2	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wor
0	290121	9053	what are your favorite quotes or sayings	what are some of your favorite quotes	1	1	41	38	7	7	4.0	14.0
1	213704	32541	what is the difference between infatuation l...	how do you discern between infatuation and love	1	1	50	48	8	8	3.0	16.0

2 rows × 230 columns

In [165]:

```
data.columns.values
```

Out[165]:

```
array(['qid1', 'qid2', 'question1', 'question2', 'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', '0_x', '1_x', '2_x',
       '3_x', '4_x', '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x',
       '12_x', '13_x', '14_x', '15_x', '16_x', '17_x', '18_x', '19_x',
       '20_x', '21_x', '22_x', '23_x', '24_x', '25_x', '26_x', '27_x',
       '28_x', '29_x', '30_x', '31_x', '32_x', '33_x', '34_x', '35_x',
       '36_x', '37_x', '38_x', '39_x', '40_x', '41_x', '42_x', '43_x',
       '44_x', '45_x', '46_x', '47_x', '48_x', '49_x', '50_x', '51_x',
       '52_x', '53_x', '54_x', '55_x', '56_x', '57_x', '58_x', '59_x',
       '60_x', '61_x', '62_x', '63_x', '64_x', '65_x', '66_x', '67_x',
       '68_x', '69_x', '70_x', '71_x', '72_x', '73_x', '74_x', '75_x',
       '76_x', '77_x', '78_x', '79_x', '80_x', '81_x', '82_x', '83_x',
       '84_x', '85_x', '86_x', '87_x', '88_x', '89_x', '90_x', '91_x',
       '92_x', '93_x', '94_x', '95_x', '96_x', '97_x', '98_x', '99_x',
       '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y', '8_y',
       '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y', '16_y',
       '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y', '24_y',
       '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y', '32_y',
       '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y', '40_y',
       '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y', '48_y',
       '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y', '56_y',
       '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y', '64_y',
       '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y', '72_y',
       '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y', '80_y',
       '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y', '88_y',
       '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y', '96_y',
       '97_y', '98_y', '99_y'], dtype=object)
```

In [166]:

```
train_data_tfidf_tr = data[[  'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', '0_x', '1_x', '2_x',
       '3_x', '4_x', '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x',
       '12_x', '13_x', '14_x', '15_x', '16_x', '17_x', '18_x', '19_x',
       '20_x', '21_x', '22_x', '23_x', '24_x', '25_x', '26_x', '27_x',
       '28_x', '29_x', '30_x', '31_x', '32_x', '33_x', '34_x', '35_x',
       '36_x', '37_x', '38_x', '39_x', '40_x', '41_x', '42_x', '43_x',
       '44_x', '45_x', '46_x', '47_x', '48_x', '49_x', '50_x', '51_x',
       '52_x', '53_x', '54_x', '55_x', '56_x', '57_x', '58_x', '59_x',
       '60_x', '61_x', '62_x', '63_x', '64_x', '65_x', '66_x', '67_x',
       '68_x', '69_x', '70_x', '71_x', '72_x', '73_x', '74_x', '75_x',
       '76_x', '77_x', '78_x', '79_x', '80_x', '81_x', '82_x', '83_x',
       '84_x', '85_x', '86_x', '87_x', '88_x', '89_x', '90_x', '91_x',
       '92_x', '93_x', '94_x', '95_x', '96_x', '97_x', '98_x', '99_x',
       '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y', '8_y',
       '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y', '16_y',
       '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y', '24_y',
       '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y', '32_y',
       '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y', '40_y',
       '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y', '48_y',
       '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y', '56_y',
       '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y', '64_y',
       '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y', '72_y',
       '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y', '80_y',
       '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y', '88_y',
       '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y', '96_y',
       '97_y', '98_y', '99_y']]
```

In [167]:

```
train_data_tfidf_tr.head(2)
```

Out[167]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1-
0	1	1	41	38	7	7	4.0	14.0	0.2857142857142857	2
1	1	1	50	48	8	8	3.0	16.0	0.1875	2

2 rows × 226 columns

In [168]:

```
train_data_tfidf_tr.shape, y_train.shape
```

Out[168]:

```
((7000, 226), (7000,))
```

8.2.2 Test data

In [169]:

```
#Creating db file from csv
import sqlalchemy
if not os.path.isfile(file_path + 'tfidf_test.db'):
  print("not present in drive")
  disk_engine = sqlalchemy.create_engine('sqlite:///'+file_path+'tfidf_test.db')
  start = dt.datetime.now()
  chunksize = 180000
  j = 0
  index_start = 1
  for df in pd.read_csv(file_path+'final_features_tfidf_test.csv',
                        names=['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
                               'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
                               'q2_n_words', 'word_Common', 'word_Total', 'word_share',
                               'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min',
                               'csc_max', 'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq',
                               'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_sort_ratio',
                               'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio',
                               '0_x', '1_x', '2_x', '3_x', '4_x',
                               '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x', '12_x', '13_x',
                               '14_x', '15_x', '16_x', '17_x', '18_x', '19_x', '20_x', '21_x',
                               '22_x', '23_x', '24_x', '25_x', '26_x', '27_x', '28_x', '29_x',
                               '30_x', '31_x', '32_x', '33_x', '34_x', '35_x', '36_x', '37_x',
                               '38_x', '39_x', '40_x', '41_x', '42_x', '43_x', '44_x', '45_x',
                               '46_x', '47_x', '48_x', '49_x', '50_x', '51_x', '52_x', '53_x',
                               '54_x', '55_x', '56_x', '57_x', '58_x', '59_x', '60_x', '61_x',
                               '62_x', '63_x', '64_x', '65_x', '66_x', '67_x', '68_x', '69_x',
                               '70_x', '71_x', '72_x', '73_x', '74_x', '75_x', '76_x', '77_x',
                               '78_x', '79_x', '80_x', '81_x', '82_x', '83_x', '84_x', '85_x',
                               '86_x', '87_x', '88_x', '89_x', '90_x', '91_x', '92_x', '93_x',
                               '94_x', '95_x', '96_x', '97_x', '98_x', '99_x',
                                '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y',
                               '7_y', '8_y', '9_y', '10_y', '11_y', '12_y', '13_y', '14_y',
                               '15_y', '16_y', '17_y', '18_y', '19_y', '20_y', '21_y', '22_y',
                               '23_y', '24_y', '25_y', '26_y', '27_y', '28_y', '29_y', '30_y',
                               '31_y', '32_y', '33_y', '34_y', '35_y', '36_y', '37_y', '38_y',
                               '39_y', '40_y', '41_y', '42_y', '43_y', '44_y', '45_y', '46_y',
                               '47_y', '48_y', '49_y', '50_y', '51_y', '52_y', '53_y', '54_y',
                               '55_y', '56_y', '57_y', '58_y', '59_y', '60_y', '61_y', '62_y',
                               '63_y', '64_y', '65_y', '66_y', '67_y', '68_y', '69_y', '70_y',
                               '71_y', '72_y', '73_y', '74_y', '75_y', '76_y', '77_y', '78_y',
                               '79_y', '80_y', '81_y', '82_y', '83_y', '84_y', '85_y', '86_y',
                               '87_y', '88_y', '89_y', '90_y', '91_y', '92_y', '93_y', '94_y',
                               '95_y', '96_y', '97_y', '98_y', '99_y'],
                        chunksize=chunksize, iterator=True, encoding='utf-8', ):
      df.index += index_start
      j+=1
      print('{} rows'.format(j*chunksize))
      df.to_sql('data', disk_engine, if_exists='append')
      index_start = df.index[-1] + 1
```

```
not present in drive
180000 rows
```

In [170]:

```
read_db = file_path + 'tfidf_test.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
data
[('data',)]
```

In [171]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)
        data =pd.read_sql_query("""SELECT * From data ORDER BY RANDOM() LIMIT 100001;""", conn_r)

        # for selecting random points
#         data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn_r)
        conn_r.commit()
        conn_r.close()
```

In [172]:

```
data.shape
```

Out[172]:

```
(3001, 233)
```

In [173]:

```
data[data['is_duplicate']=='is_duplicate']
```

Out[173]:

	index	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
2443	NaN	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words

1 rows × 233 columns

In [174]:

```
data = data.drop(data.index[data['is_duplicate']=='is_duplicate'])
```

In [175]:

```
data.shape
```

Out[175]:

```
(3000, 233)
```

In [176]:

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

In [177]:

```
data.shape, y_test.shape
```

Out[177]:

```
((3000, 230), (3000,))
```

In [178]:

```
data.head(2)
```

Out[178]:

	qid1	qid2	question1	question2	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wo
0	84277	123591	why even though my nose is oily the skin kee...	what causes skin to peel off	1	1	86	29	17	6	3.0	23.
1	394136	394137	what are the best law colleges in india	which is the best college for law in india	1	1	40	43	8	9	5.0	17.

2 rows × 230 columns

In [179]:

```
data.columns.values
```

Out[179]:

```
array(['qid1', 'qid2', 'question1', 'question2', 'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', '0_x', '1_x', '2_x',
       '3_x', '4_x', '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x',
       '12_x', '13_x', '14_x', '15_x', '16_x', '17_x', '18_x', '19_x',
       '20_x', '21_x', '22_x', '23_x', '24_x', '25_x', '26_x', '27_x',
       '28_x', '29_x', '30_x', '31_x', '32_x', '33_x', '34_x', '35_x',
       '36_x', '37_x', '38_x', '39_x', '40_x', '41_x', '42_x', '43_x',
       '44_x', '45_x', '46_x', '47_x', '48_x', '49_x', '50_x', '51_x',
       '52_x', '53_x', '54_x', '55_x', '56_x', '57_x', '58_x', '59_x',
       '60_x', '61_x', '62_x', '63_x', '64_x', '65_x', '66_x', '67_x',
       '68_x', '69_x', '70_x', '71_x', '72_x', '73_x', '74_x', '75_x',
       '76_x', '77_x', '78_x', '79_x', '80_x', '81_x', '82_x', '83_x',
       '84_x', '85_x', '86_x', '87_x', '88_x', '89_x', '90_x', '91_x',
       '92_x', '93_x', '94_x', '95_x', '96_x', '97_x', '98_x', '99_x',
       '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y', '8_y',
       '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y', '16_y',
       '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y', '24_y',
       '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y', '32_y',
       '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y', '40_y',
       '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y', '48_y',
       '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y', '56_y',
       '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y', '64_y',
       '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y', '72_y',
       '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y', '80_y',
       '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y', '88_y',
       '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y', '96_y',
       '97_y', '98_y', '99_y'], dtype=object)
```

In [180]:

```
test_data_tfidf_test = data[[  'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'cwc_min',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', '0_x', '1_x', '2_x',
       '3_x', '4_x', '5_x', '6_x', '7_x', '8_x', '9_x', '10_x', '11_x',
       '12_x', '13_x', '14_x', '15_x', '16_x', '17_x', '18_x', '19_x',
       '20_x', '21_x', '22_x', '23_x', '24_x', '25_x', '26_x', '27_x',
       '28_x', '29_x', '30_x', '31_x', '32_x', '33_x', '34_x', '35_x',
       '36_x', '37_x', '38_x', '39_x', '40_x', '41_x', '42_x', '43_x',
       '44_x', '45_x', '46_x', '47_x', '48_x', '49_x', '50_x', '51_x',
       '52_x', '53_x', '54_x', '55_x', '56_x', '57_x', '58_x', '59_x',
       '60_x', '61_x', '62_x', '63_x', '64_x', '65_x', '66_x', '67_x',
       '68_x', '69_x', '70_x', '71_x', '72_x', '73_x', '74_x', '75_x',
       '76_x', '77_x', '78_x', '79_x', '80_x', '81_x', '82_x', '83_x',
       '84_x', '85_x', '86_x', '87_x', '88_x', '89_x', '90_x', '91_x',
       '92_x', '93_x', '94_x', '95_x', '96_x', '97_x', '98_x', '99_x',
       '0_y', '1_y', '2_y', '3_y', '4_y', '5_y', '6_y', '7_y', '8_y',
       '9_y', '10_y', '11_y', '12_y', '13_y', '14_y', '15_y', '16_y',
       '17_y', '18_y', '19_y', '20_y', '21_y', '22_y', '23_y', '24_y',
       '25_y', '26_y', '27_y', '28_y', '29_y', '30_y', '31_y', '32_y',
       '33_y', '34_y', '35_y', '36_y', '37_y', '38_y', '39_y', '40_y',
       '41_y', '42_y', '43_y', '44_y', '45_y', '46_y', '47_y', '48_y',
       '49_y', '50_y', '51_y', '52_y', '53_y', '54_y', '55_y', '56_y',
       '57_y', '58_y', '59_y', '60_y', '61_y', '62_y', '63_y', '64_y',
       '65_y', '66_y', '67_y', '68_y', '69_y', '70_y', '71_y', '72_y',
       '73_y', '74_y', '75_y', '76_y', '77_y', '78_y', '79_y', '80_y',
       '81_y', '82_y', '83_y', '84_y', '85_y', '86_y', '87_y', '88_y',
       '89_y', '90_y', '91_y', '92_y', '93_y', '94_y', '95_y', '96_y',
       '97_y', '98_y', '99_y']]
```

In [181]:

```
test_data_tfidf_test.head(2)
```

Out[181]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q
0	1	1	86	29	17	6	3.0	23.0	0.13043478260869565	2
1	1	1	40	43	8	9	5.0	17.0	0.29411764705882354	2

2 rows × 226 columns

In [182]:

```
test_data_tfidf_test.shape, y_test.shape
```

Out[182]:

```
((3000, 226), (3000,))
```

8.2.3 Converting String to Numerics

In [183]:

```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
# https://stackoverflow.com/questions/40790031/pandas-to-numeric-find-out-which-string-it-was-unab
le-to-parse
cols = list(train_data_tfidf_tr.columns)
for i in cols:
    train_data_tfidf_tr[i] = train_data_tfidf_tr[i].apply(pd.to_numeric, errors='coerce')
    print(i)
```

```
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
cwc_min
cwc_max
csc_min
```

```
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
0_x
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
10_x
11_x
12_x
13_x
14_x
15_x
16_x
17_x
18_x
19_x
20_x
21_x
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
39_x
40_x
41_x
42_x
43_x
44_x
45_x
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
```

```
65_x
66_x
67_x
68_x
69_x
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
96_x
97_x
98_x
99_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
22_y
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
40_y
41_y
```

```
41_y
42_y
43_y
44_y
45_y
46_y
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
96_y
97_y
98_y
99_y
```

In [184]:

```
y_train[:10]
```

Out[184]:

```
array(['1', '1', '0', '1', '0', '0', '0', '0', '0', '0'], dtype=object)
```

In [185]:

```
y_train = pd.Series(map(int,list(y_train)))
```

In [186]:

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

```
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
0_x
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
10_x
11_x
12_x
13_x
14_x
15_x
16_x
17_x
18_x
19_x
20_x
21_x
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
39_x
40_x
41_x
42_x
43_x
44_x
45_x
```

```
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
65_x
66_x
67_x
68_x
69_x
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
96_x
97_x
98_x
99_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
22_y
```

```
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
40_y
41_y
42_y
43_y
44_y
45_y
46_y
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
96_y
97_y
98_y
99_y
```

In [187]:

```
y_test[1:10]
```

Out[187]:

```
array(['1', '0', '0', '1', '1', '1', '1', '0', '0'], dtype=object)
```

In [188]:

```
y_test = pd.Series(map(int,list(y_test)))
```

In [189]:

```
x_tr_set2 = train_data_tfidf_tr
x_test_set2 = test_data_tfidf_test
y_tr_set2 = y_train
y_test_set2 = y_test
```

In [190]:

```
y_test
```

Out[190]:

```
0       0
1       1
2       0
3       0
4       1
       ..
2995    0
2996    1
2997    0
2998    1
2999    0
Length: 3000, dtype: int64
```

In [191]:

```
x_tr_set1.shape, x_test_set1.shape, y_tr_set1.shape, y_test_set1.shape
```

Out[191]:

```
((7000, 218), (3000, 218), (7000,), (3000,))
```

In [192]:

```
x_tr_set2.shape, x_test_set2.shape, y_tr_set2.shape, y_test_set2.shape
```

Out[192]:

```
((7000, 226), (3000, 226), (7000,), (3000,))
```

9.Models

In [193]:

```
# 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],
    # 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()
```

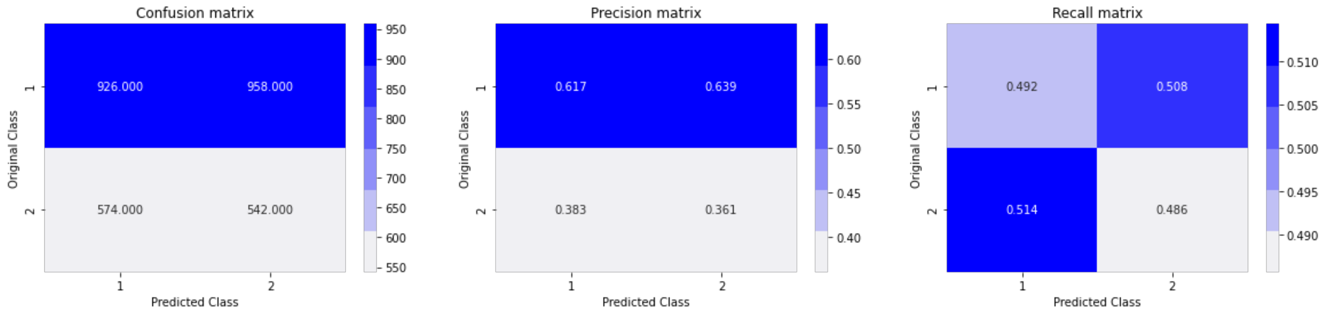
9.1 Random model

9.1.1 on Set1 TFIDF-W2V

In [194]:

```
# 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

train_len = len(y_tr_set1)
predicted_y = np.zeros((train_len,2))
for i in range(train_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])

loss_tr = log_loss(y_tr_set1, predicted_y, eps=1e-15)
print("Log loss on Train Data using Random Model",log_loss(y_tr_set1, predicted_y, eps=1e-15))

test_len = len(y_test_set1)
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])
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])

loss_test = log_loss(y_test_set1, predicted_y, eps=1e-15)
print("Log loss on Test Data using Random Model",log_loss(y_test_set1, predicted_y, eps=1e-15))

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

```
Log loss on Train Data using Random Model 0.8909857413369713
Log loss on Test Data using Random Model 0.8839580037167362
```



In [195]:

```
from prettytable import PrettyTable
summary = PrettyTable()
summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log loss on Test"]
summary.add_row(['TFIDF_w2v','Random dumb model', "-" , loss_tr, loss_test ])
print(summary)
```

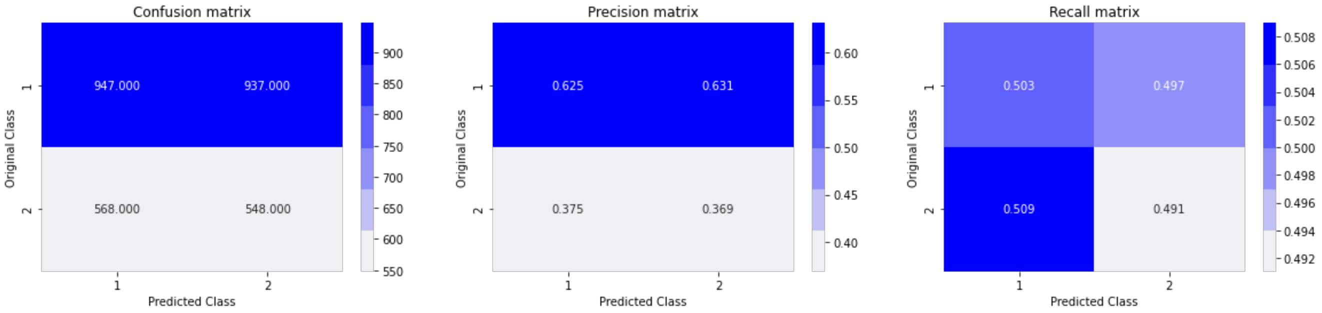
```
+------------+-------------------+----------------------+--------------------+--------------------+
| Vectorizer |       Model       | Best Hyperparameters | Log loss on Train  |  Log loss on Test  |
+------------+-------------------+----------------------+--------------------+--------------------+
| TFIDF_w2v  | Random dumb model |          -           | 0.8909857413369713 | 0.8839580037167362 |
+------------+-------------------+----------------------+--------------------+--------------------+
```

9.1.2 on Set2 TFIDF

In [196]:

```
# 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

train_len = len(y_tr_set2)
predicted_y = np.zeros((train_len,2))
for i in range(train_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])

loss_tr = log_loss(y_tr_set2, predicted_y, eps=1e-15)
print("Log loss on Train Data using Random Model",log_loss(y_tr_set2, predicted_y, eps=1e-15))

test_len = len(y_test_set2)
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])

loss_test = log_loss(y_test_set2, predicted_y, eps=1e-15)
print("Log loss on Test Data using Random Model",log_loss(y_test_set2, predicted_y, eps=1e-15))

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

```
Log loss on Train Data using Random Model 0.8681386222435589
Log loss on Test Data using Random Model 0.9077642168454928
```



In [197]:

```
# from prettytable import PrettyTable
# summary = PrettyTable()
summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF','Random dumb model', "-" , loss_tr, loss_test ])
print(summary)
```

```
+------------+-------------------+----------------------+--------------------+--------------------+
| Vectorizer |       Model       | Best Hyperparameters | Log loss on Train  |  Log loss on Test  |
+------------+-------------------+----------------------+--------------------+--------------------+
| TFIDF_w2v  | Random dumb model |          -           | 0.8909857413369713 | 0.8839580037167362 |
|   TFIDF    | Random dumb model |          -           | 0.8681386222435589 | 0.9077642168454928 |
+------------+-------------------+----------------------+--------------------+--------------------+
```

9.2 Logistic Regression with hyperparameter tuning

9.2.1 on Set1 TFIDF-W2V

In [198]:

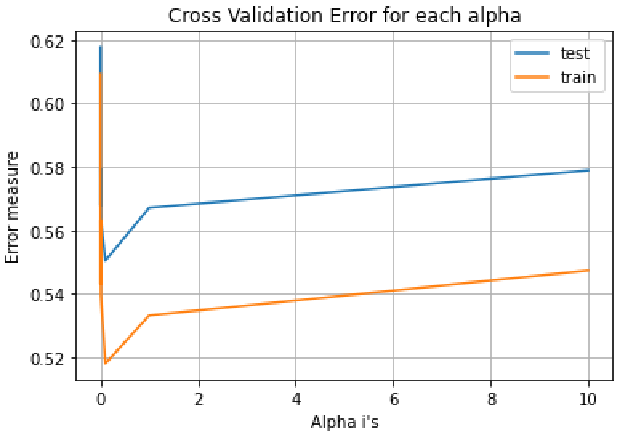
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_test = []
log_error_tr = []

for c,i in enumerate(alpha):
  clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
  clf.fit(x_tr_set1, y_tr_set1)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr_set1, y_tr_set1)

  predict_y = sig_clf.predict_proba(x_tr_set1)
  log_error_tr.append(log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15))

  predict_y = sig_clf.predict_proba(x_test_set1)
  log_error_test.append(log_loss(y_test_set1, predict_y, labels=clf.classes_, eps=1e-15))
  print("="*40)
  print('values of alpha = ', i)
```

```
    print( "values of alpha = ", i)
    print( "The Train log loss is:", log_error_tr[c])
    print( "The Test log loss is:", log_error_test[c])
```

```
========================================
values of alpha =  1e-05
The Train log loss is: 0.6091916455448889
The Test log loss is: 0.6176956398711984
========================================
values of alpha =  0.0001
The Train log loss is: 0.543096536913335
The Test log loss is: 0.5677820606079191
========================================
values of alpha =  0.001
The Train log loss is: 0.5633423389191436
The Test log loss is: 0.5794821211753041
========================================
values of alpha =  0.01
The Train log loss is: 0.5394195102762249
The Test log loss is: 0.5632093221739694
========================================
values of alpha =  0.1
The Train log loss is: 0.5181022850363174
The Test log loss is: 0.5504720745790216
========================================
values of alpha =  1
The Train log loss is: 0.5332981263234295
The Test log loss is: 0.567115087691264
========================================
values of alpha =  10
The Train log loss is: 0.5473516900016429
The Test log loss is: 0.5788983703201077
```

In [199]:

```
fig, ax = plt.subplots()
ax.plot(alpha, log_error_test, label="test")
# 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]))

ax.plot(alpha, log_error_tr, label="train")
# 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.legend()
plt.show()
```



In [200]:

```
best_alpha = np.argmin(log_error_test)
alpha[best_alpha]
```

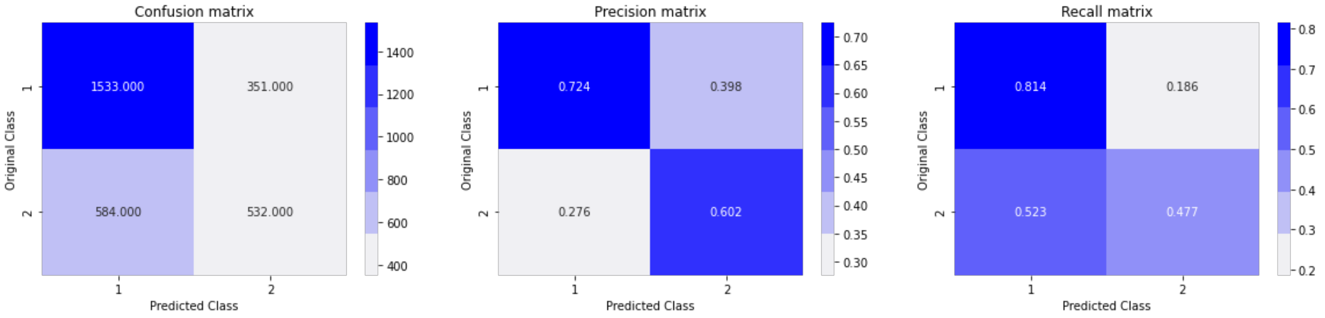
Out[200]:

```
0.1
```

In [201]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(x_tr_set1, y_tr_set1)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr_set1, y_tr_set1)

predict_y = sig_clf.predict_proba(x_tr_set1)
loss_tr = log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(x_test_set1)
loss_test = log_loss(y_test_set1, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_set1, 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_set1, predicted_y)
```

```
For values of best alpha =  0.1 The train log loss is: 0.5181022850363174
For values of best alpha =  0.1 The test log loss is: 0.5504720745790216
Total number of data points : 3000
```



In [202]:

```
# summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF_w2v', 'Logistic Regression by iterative search', "alpha = " +
str(alpha[best_alpha]) , loss_tr, loss_test ])
print(summary)
```

```
+------------+------------------------------------------+----------------------+--------------------
-------------------+
| Vectorizer |                  Model                   | Best Hyperparameters | Log loss on Train
Log loss on Test   |
+------------+------------------------------------------+----------------------+--------------------
-------------------+
| TFIDF_w2v  |            Random dumb model             |          -           | 0.8909857413369713
0.8839580037167362 |
|   TFIDF    |            Random dumb model             |          -           | 0.8681386222435589
0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search  |     alpha = 0.1      | 0.5181022850363174
| 0.5504720745790216 |
+------------+------------------------------------------+----------------------+--------------------
-------------------+
```

9.2.2 on Set2 TFIDF

In [203]:

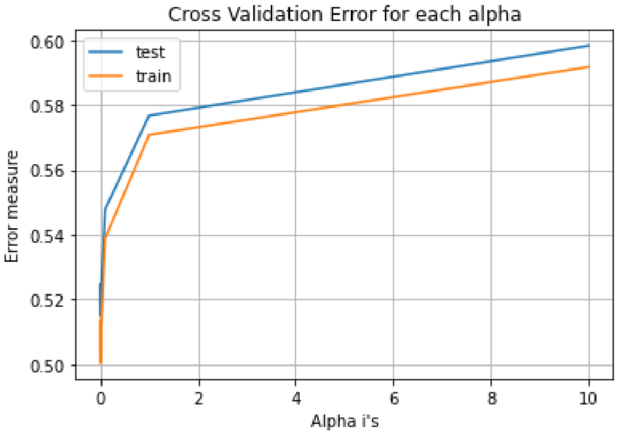
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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_test = []
log_error_tr = []

for c,i in enumerate(alpha):
  clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
  clf.fit(x_tr_set2, y_tr_set2)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr_set2, y_tr_set2)

  predict_y = sig_clf.predict_proba(x_tr_set2)
  log_error_tr.append(log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15))

  predict_y = sig_clf.predict_proba(x_test_set2)
  log_error_test.append(log_loss(y_test_set2, predict_y, labels=clf.classes_, eps=1e-15))
  print("="*40)
  print('values of alpha = ', i)
  print( "The Train log loss is:", log_error_tr[c])
  print( "The Test log loss is:", log_error_test[c])
```

```
========================================
values of alpha =  1e-05
The Train log loss is: 0.5033447327977969
The Test log loss is: 0.518974269660145
========================================
values of alpha =  0.0001
The Train log loss is: 0.5136749062267901
The Test log loss is: 0.5246955453491132
========================================
values of alpha =  0.001
The Train log loss is: 0.5004491052297945
The Test log loss is: 0.5149092374073378
========================================
values of alpha =  0.01
The Train log loss is: 0.507871340132494
The Test log loss is: 0.5198158058284845
========================================
values of alpha =  0.1
The Train log loss is: 0.538881494767776
The Test log loss is: 0.547810219482023
========================================
values of alpha =  1
The Train log loss is: 0.5708097348310461
The Test log loss is: 0.576765884165689
========================================
values of alpha =  10
The Train log loss is: 0.5917558991396166
The Test log loss is: 0.598314345312015
```

In [204]:

```
fig, ax = plt.subplots()
ax.plot(alpha, log_error_test, label="test")
# 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]))

ax.plot(alpha, log_error_tr, label="train")
# 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]))
#      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.legend()
plt.show()
```



In [205]:

```
best_alpha = np.argmin(log_error_test)
alpha[best_alpha]
```

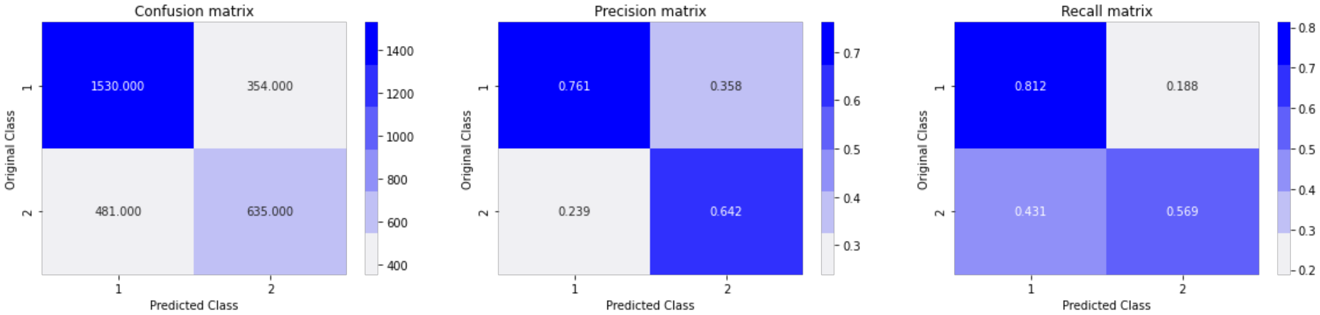
Out[205]:

```
0.001
```

In [206]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(x_tr_set2, y_tr_set2)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr_set2, y_tr_set2)

predict_y = sig_clf.predict_proba(x_tr_set2)
loss_tr = log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(x_test_set2)
loss_test = log_loss(y_test_set2, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_set2, 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_set2, predicted_y)
```

```
For values of best alpha =  0.001 The train log loss is: 0.5004491052297945
For values of best alpha =  0.001 The test log loss is: 0.5149092374073378
Total number of data points : 3000
```



In [207]:

```
# summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF', 'Logistic Regression by iterative search', "alpha = " +
str(alpha[best_alpha]) , loss_tr, loss_test ])
print(summary)
```

```
+------------+-----------------------------------------+----------------------+--------------------
------------------+
| Vectorizer |                  Model                  | Best Hyperparameters | Log loss on Train
Log loss on Test  |
+------------+-----------------------------------------+----------------------+--------------------
------------------+
| TFIDF_w2v  |            Random dumb model            |          -           | 0.8909857413369713
0.8839580037167362 |
|   TFIDF    |            Random dumb model            |          -           | 0.8681386222435589
0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search |     alpha = 0.1      | 0.5181022850363174
| 0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |    alpha = 0.001     | 0.5004491052297945
| 0.5149092374073378 |
+------------+-----------------------------------------+----------------------+--------------------
------------------+
```

9.3 Linear SVM with hyperparameter tuning

9.3.1 on Set1 TFIDF-W2V

9.3.1.1 using L1 regularizer

In [208]:

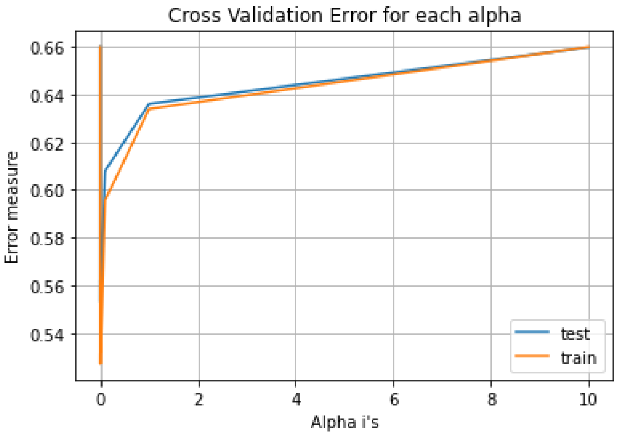
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_test = []
log_error_tr = []

for c,i in enumerate(alpha):
  clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
  clf.fit(x_tr_set1, y_tr_set1)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr_set1, y_tr_set1)

  predict_y = sig_clf.predict_proba(x_tr_set1)
  log_error_tr.append(log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15))

  predict_y = sig_clf.predict_proba(x_test_set1)
  log_error_test.append(log_loss(y_test_set1, predict_y, labels=clf.classes_, eps=1e-15))
  print("="*40)
  print('values of alpha = ', i)
  print( "The Train log loss is:", log_error_tr[c])
```

```
    print( "The Test log loss is:", log_error_test[c])
```

```
========================================
values of alpha =  1e-05
The Train log loss is: 0.6599367618737666
The Test log loss is: 0.660011544065058
========================================
values of alpha =  0.0001
The Train log loss is: 0.527068330818554
The Test log loss is: 0.5528737395236669
========================================
values of alpha =  0.001
The Train log loss is: 0.5312839392947547
The Test log loss is: 0.5620421712320395
========================================
values of alpha =  0.01
The Train log loss is: 0.5331349687886794
The Test log loss is: 0.558945396873255
========================================
values of alpha =  0.1
The Train log loss is: 0.5956886197710739
The Test log loss is: 0.6080006713561955
========================================
values of alpha =  1
The Train log loss is: 0.6339287504368991
The Test log loss is: 0.6360698125216653
========================================
values of alpha =  10
The Train log loss is: 0.659919827255779
The Test log loss is: 0.6597429897754892
```

In [209]:

```
fig, ax = plt.subplots()
ax.plot(alpha, log_error_test, label="test")
# 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]))

ax.plot(alpha, log_error_tr, label="train")
# 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.legend()
plt.show()
```



In [210]:

```
best_alpha = np.argmin(log_error_test)
alpha[best_alpha]
```

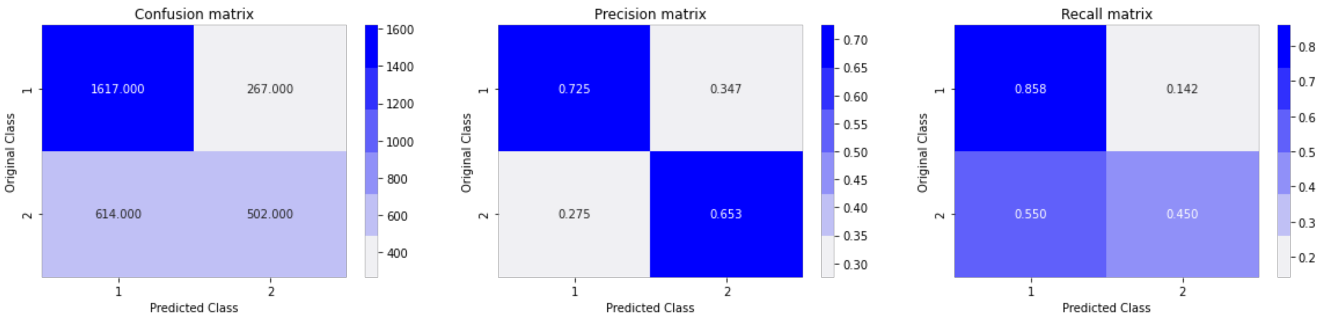
Out[210]:

```
0.0001
```

In [211]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(x_tr_set1, y_tr_set1)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr_set1, y_tr_set1)

predict_y = sig_clf.predict_proba(x_tr_set1)
loss_tr = log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(x_test_set1)
loss_test = log_loss(y_test_set1, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_set1, 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_set1, predicted_y)
```

```
For values of best alpha =  0.0001 The train log loss is: 0.527068330818554
For values of best alpha =  0.0001 The test log loss is: 0.5528737395236669
Total number of data points : 3000
```



In [212]:

```
# summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF_w2v', 'Linear SVM by iterative search', "L1 Norm & alpha = " + str(alpha[b
est_alpha]) , loss_tr, loss_test ])
print(summary)
```

```
+------------+-----------------------------------------+--------------------------+----------------
+--------------------+
| Vectorizer |                  Model                  |   Best Hyperparameters   | Log loss on Tra
n  |  Log loss on Test  |
+------------+-----------------------------------------+--------------------------+----------------
+--------------------+
| TFIDF_w2v  |             Random dumb model           |            -             |
0.8909857413369713 | 0.8839580037167362 |
|   TFIDF    |             Random dumb model           |            -             | 0.8681386222435
9 | 0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search |       alpha = 0.1        |
0.5181022850363174 | 0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |      alpha = 0.001       |
0.5004491052297945 | 0.5149092374073378 |
| TFIDF_w2v  |      Linear SVM by iterative search     | L1 Norm & alpha = 0.0001 | 0.5270683308185
54   | 0.5528737395236669 |
+------------+-----------------------------------------+--------------------------+----------------
+--------------------+
```

9.3.1.2 using L2 regularizer

In [213]:

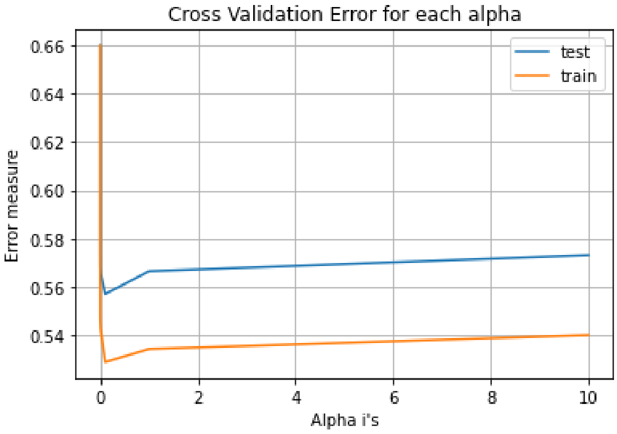
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier,
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_test = []
log_error_tr = []

for c,i in enumerate(alpha):
  clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
  clf.fit(x_tr_set1, y_tr_set1)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr_set1, y_tr_set1)

  predict_y = sig_clf.predict_proba(x_tr_set1)
  log_error_tr.append(log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15))

  predict_y = sig_clf.predict_proba(x_test_set1)
  log_error_test.append(log_loss(y_test_set1, predict_y, labels=clf.classes_, eps=1e-15))
  print("="*40)
  print('values of alpha = ', i)
  print( "The Train log loss is:", log_error_tr[c])
  print( "The Test log loss is:", log_error_test[c])
```

```
========================================
values of alpha =  1e-05
The Train log loss is: 0.6599367618737666
The Test log loss is: 0.660011544065058
========================================
values of alpha =  0.0001
The Train log loss is: 0.5483462114713233
The Test log loss is: 0.5734025274118046
========================================
values of alpha =  0.001
The Train log loss is: 0.5463185742896588
The Test log loss is: 0.5680875320477938
========================================
values of alpha =  0.01
The Train log loss is: 0.5422897955597696
The Test log loss is: 0.5653708871224005
========================================
values of alpha =  0.1
The Train log loss is: 0.5287501978376552
The Test log loss is: 0.5569509511703374
========================================
values of alpha =  1
The Train log loss is: 0.5341690990494615
The Test log loss is: 0.5664107320888319
========================================
values of alpha =  10
The Train log loss is: 0.5399361664670905
The Test log loss is: 0.5730685406280625
```

In [214]:

```
fig, ax = plt.subplots()
ax.plot(alpha, log_error_test, label="test")
# 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]))
```

```
ax.plot(alpha, log_error_tr, label="train")
# 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.legend()
plt.show()
```



In [215]:

```
best_alpha = np.argmin(log_error_test)
alpha[best_alpha]
```

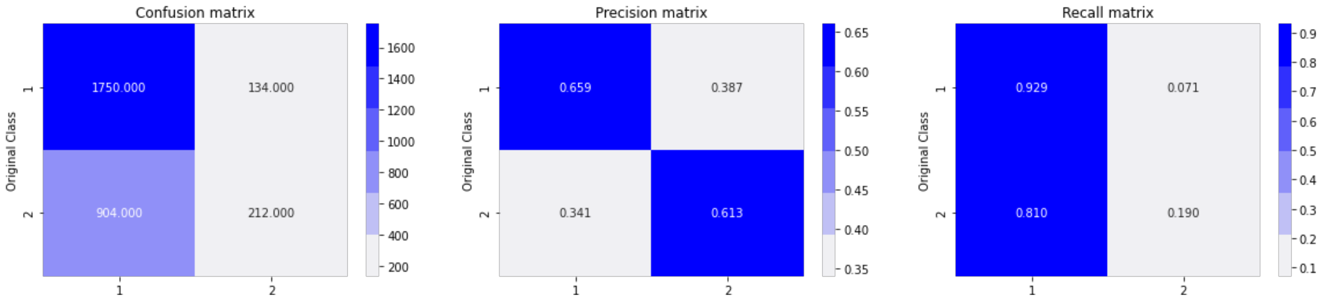
Out[215]:

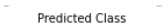
```
0.1
```

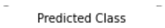
In [216]:

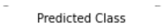
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(x_tr_set1, y_tr_set1)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr_set1, y_tr_set1)

predict_y = sig_clf.predict_proba(x_tr_set1)
loss_tr = log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_tr_set1, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(x_test_set1)
loss_test = log_loss(y_test_set1, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test_set1, 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_set1, predicted_y)
```

```
For values of best alpha =  0.1 The train log loss is: 0.5956886197710739
For values of best alpha =  0.1 The test log loss is: 0.6080006713561955
Total number of data points : 3000
```









In [217]:

```
# summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF_w2v', 'Linear SVM by iterative search', "L2 Norm & alpha = " + str(alpha[b
est_alpha]) , loss_tr, loss_test ])
print(summary)
```

```
+------------+----------------------------------------+--------------------------+----------------
+--------------------+
| Vectorizer |                 Model                  |   Best Hyperparameters   | Log loss on Tra
n  |  Log loss on Test  |
+------------+----------------------------------------+--------------------------+----------------
+--------------------+
| TFIDF_w2v  |            Random dumb model           |            -             |
0.8909857413369713 | 0.8839580037167362 |
|   TFIDF    |            Random dumb model           |            -             | 0.8681386222435
9 | 0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search |       alpha = 0.1        |
0.5181022850363174 | 0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |      alpha = 0.001       |
0.5004491052297945 | 0.5149092374073378 |
| TFIDF_w2v  |      Linear SVM by iterative search     | L1 Norm & alpha = 0.0001 | 0.5270683308185
54  | 0.5528737395236669 |
| TFIDF_w2v  |      Linear SVM by iterative search     |  L2 Norm & alpha = 0.1   | 0.595688619771C
739 | 0.6080006713561955 |
+------------+----------------------------------------+--------------------------+----------------
+--------------------+
```

9.3.2 on Set2 TFIDF

9.3.2.1 using L1 regularizer

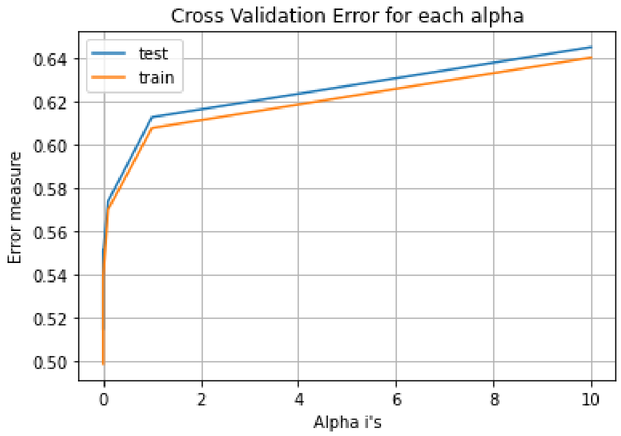
In [218]:

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

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

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

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


log_error_test = []
log_error_tr = []

for c,i in enumerate(alpha):
  clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
  clf.fit(x_tr_set2, y_tr_set2)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr_set2, y_tr_set2)

  predict_y = sig_clf.predict_proba(x_tr_set2)
  log_error_tr.append(log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15))

  predict_y = sig_clf.predict_proba(x_test_set2)
  log_error_test.append(log_loss(y_test_set2, predict_y, labels=clf.classes_, eps=1e-15))
  print("="*40)
  print('values of alpha = ', i)
  print( "The Train log loss is:", log_error_tr[c])
  print( "The Test log loss is:", log_error_test[c])
```

```
========================================
values of alpha =  1e-05
The Train log loss is: 0.49838297495456246
The Test log loss is: 0.5148041636287534
========================================
values of alpha =  0.0001
The Train log loss is: 0.5304182143472248
The Test log loss is: 0.544454054175011
========================================
values of alpha =  0.001
The Train log loss is: 0.5389807644172705
The Test log loss is: 0.551159201065387
========================================
values of alpha =  0.01
The Train log loss is: 0.5418008693703278
The Test log loss is: 0.5509921907365074
========================================
values of alpha =  0.1
The Train log loss is: 0.5699004693783398
The Test log loss is: 0.573834749185796
========================================
values of alpha =  1
The Train log loss is: 0.6077947237022424
The Test log loss is: 0.6128287980616041
========================================
values of alpha =  10
The Train log loss is: 0.6404419011995035
The Test log loss is: 0.6452703803045354
```

In [219]:

```
fig, ax = plt.subplots()
ax.plot(alpha, log_error_test, label="test")
# 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]))

ax.plot(alpha, log_error_tr, label="train")
# 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.legend()
plt.show()
```



In [220]:

```
best_alpha = np.argmin(log_error_test)
alpha[best_alpha]
```

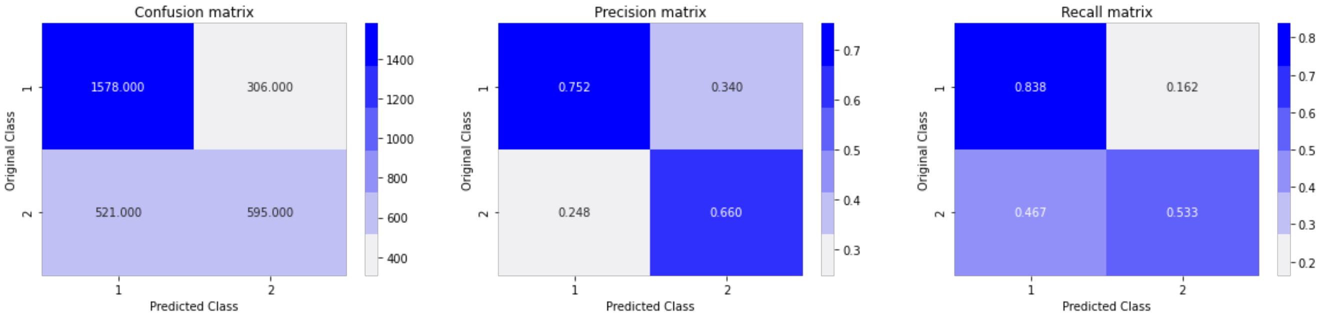
Out[220]:

```
1e-05
```

In [221]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(x_tr_set2, y_tr_set2)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr_set2, y_tr_set2)

predict_y = sig_clf.predict_proba(x_tr_set2)
loss_tr = log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(x_test_set2)
loss_test = log_loss(y_test_set2, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_set2, 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_set2, predicted_y)
```

```
For values of best alpha =  1e-05 The train log loss is: 0.49838297495456246
For values of best alpha =  1e-05 The test log loss is: 0.5148041636287534
Total number of data points : 3000
```



In [222]:

```
# summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF', 'Linear SVM by iterative search', "L1 Norm & alpha = " +
str(alpha[best_alpha]) , loss_tr, loss_test ])
print(summary)
```

```
+------------+-----------------------------------------+--------------------------+----------------
-+---------------------+
| Vectorizer |                  Model                  |   Best Hyperparameters   |  Log loss on Tr
in   |  Log loss on Test   |
+------------+-----------------------------------------+--------------------------+----------------
-+---------------------+
| TFIDF_w2v  |             Random dumb model           |            -             |
0.8909857413369713 | 0.8839580037167362 |
|   TFIDF    |             Random dumb model           |            -             |  0.868138622243
89 | 0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search |       alpha = 0.1        |
0.5181022850363174 | 0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |      alpha = 0.001       |
0.5004491052297945 | 0.5149092374073378 |
| TFIDF_w2v  |       Linear SVM by iterative search      | L1 Norm & alpha = 0.0001 |  0.527068330818
554  | 0.5528737395236669 |
| TFIDF_w2v  |       Linear SVM by iterative search      |  L2 Norm & alpha = 0.1   |  0.595688619771
0739 | 0.6080006713561955 |
|   TFIDF    |       Linear SVM by iterative search      | L1 Norm & alpha = 1e-05  |
0.49838297495456246 | 0.5148041636287534 |
+------------+-----------------------------------------+--------------------------+----------------
-+---------------------+
```

9.3.2.2 using L2 regularizer

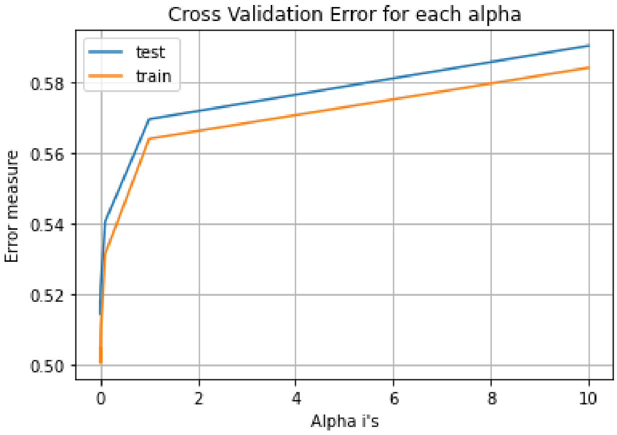
In [223]:

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

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

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

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


log_error_test = []
log_error_tr = []

for c,i in enumerate(alpha):
  clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
  clf.fit(x_tr_set2, y_tr_set2)
  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
  sig_clf.fit(x_tr_set2, y_tr_set2)

  predict_y = sig_clf.predict_proba(x_tr_set2)
  log_error_tr.append(log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15))

  predict_y = sig_clf.predict_proba(x_test_set2)
  log_error_test.append(log_loss(y_test_set2, predict_y, labels=clf.classes_, eps=1e-15))
  print("="*40)
  print('values of alpha = ', i)
  print( "The Train log loss is:", log_error_tr[c])
  print( "The Test log loss is:", log_error_test[c])
```

```
========================================
values of alpha =  1e-05
The Train log loss is: 0.5047591525799148
The Test log loss is: 0.5180016919378869
========================================
values of alpha =  0.0001
The Train log loss is: 0.5013833062480888
The Test log loss is: 0.5145545037573294
========================================
values of alpha =  0.001
The Train log loss is: 0.5006735023574405
The Test log loss is: 0.51441888795156
========================================
values of alpha =  0.01
The Train log loss is: 0.5098581958138237
The Test log loss is: 0.521919486355566
========================================
values of alpha =  0.1
The Train log loss is: 0.5314422133355698
The Test log loss is: 0.540589236217171
========================================
values of alpha =  1
The Train log loss is: 0.5639767546292684
The Test log loss is: 0.5695336195940539
========================================
values of alpha =  10
The Train log loss is: 0.5840678011399411
The Test log loss is: 0.5902832959597724
```

In [224]:

```
fig, ax = plt.subplots()
ax.plot(alpha, log_error_test, label="test")
# 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]))

ax.plot(alpha, log_error_tr, label="train")
# 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.legend()
plt.show()
```



In [225]:

```
best_alpha = np.argmin(log_error_test)
alpha[best_alpha]
```

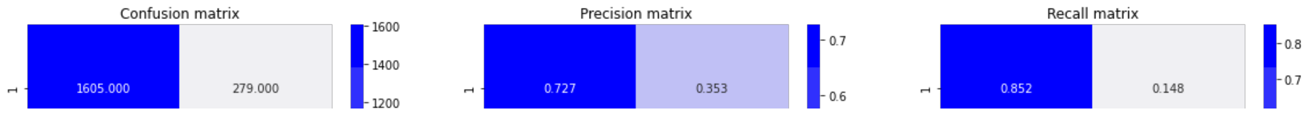
Out[225]:

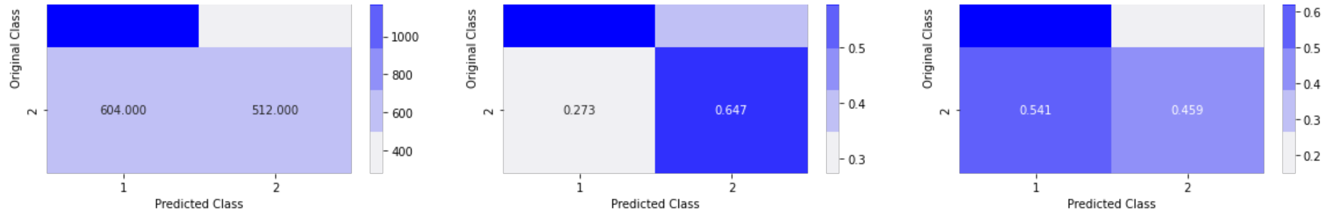
```
0.001
```

In [226]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(x_tr_set2, y_tr_set2)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_tr_set2, y_tr_set2)

predict_y = sig_clf.predict_proba(x_tr_set2)
loss_tr = log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_tr_set2, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(x_test_set2)
loss_test = log_loss(y_test_set2, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_set2, 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_set2, predicted_y)
```

```
For values of best alpha =  0.001 The train log loss is: 0.5389807644172705
For values of best alpha =  0.001 The test log loss is: 0.551159201065387
Total number of data points : 3000
```





In [227]:

```
# summary.field_names = ["Vectorizer", "Model", "Best Hyperparameters", "Log loss on Train", "Log
loss on Test"]
summary.add_row(['TFIDF', 'Linear SVM by iterative search', "L2 Norm & alpha = " +
str(alpha[best_alpha]) , loss_tr, loss_test ])
print(summary)
```

```
+------------+------------------------------------------+--------------------------+----------------
-+---------------------+
| Vectorizer |                  Model                   |   Best Hyperparameters   |  Log loss on Tr
in   |  Log loss on Test   |
+------------+------------------------------------------+--------------------------+----------------
-+---------------------+
| TFIDF_w2v  |             Random dumb model            |            -             |
0.8909857413369713 | 0.8839580037167362 |
|   TFIDF    |             Random dumb model            |            -             |  0.868138622243
89 | 0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search |       alpha = 0.1        |
0.5181022850363174 | 0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |      alpha = 0.001       |
0.5004491052297945 | 0.5149092374073378 |
| TFIDF_w2v  |       Linear SVM by iterative search     | L1 Norm & alpha = 0.0001 |  0.527068330818
554  | 0.5528737395236669 |
| TFIDF_w2v  |       Linear SVM by iterative search     |  L2 Norm & alpha = 0.1   |  0.595688619771
0739 | 0.6080006713561955 |
|   TFIDF    |       Linear SVM by iterative search     | L1 Norm & alpha = 1e-05  |
0.49838297495456246 | 0.5148041636287534 |
|   TFIDF    |       Linear SVM by iterative search     | L2 Norm & alpha = 0.001  |
0.5389807644172705 | 0.551159201065387   |
+------------+------------------------------------------+--------------------------+----------------
-+---------------------+
```

9.4 XGBoost

9.4.1 on set1 TFIDF-W2V

In [228]:

```
# https://www.kaggle.com/phunter/xgboost-with-gridsearchcv
xgb_clf = xgb.XGBClassifier()

parameters = {'objective' : ['binary:logistic'],  'eval_metric' : ['logloss'],
        'max_depth': [1,2,3,4],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
         }
d_train = xgb.DMatrix(x_tr_set1, label=y_tr_set1)
d_test = xgb.DMatrix(x_test_set1, label=y_test_set1)

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

In [229]:

```
clf = GridSearchCV(xgb_clf, parameters, n_jobs = -1,
                   cv=3,
                   scoring='neg_log_loss',
                   verbose=2, refit=True)
```

In [230]:

```
clf.fit(x_tr_set1, y_tr_set1)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:  4.4min finished
```

Out[230]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=-1,
             param_grid={'eval_metric': ['logloss'],
                         'learning_rate': [0.001, 0.01, 0.1, 0.2, 0, 3],
                         'max_depth': [1, 2, 3, 4],
                         'objective': ['binary:logistic']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_log_loss', verbose=2)
```

In [231]:

```python
for param_name in sorted(clf.best_params_.keys()):
    print((param_name, clf.best_params_[param_name]))
```

```
('eval_metric', 'logloss')
('learning_rate', 0.1)
('max_depth', 4)
('objective', 'binary:logistic')
```

In [232]:

```python
best_lr = clf.best_params_['learning_rate']
best_max_depth = clf.best_params_['max_depth']
```

In [233]:

```python
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['learning_rate'] = best_lr
params['max_depth'] = best_max_depth

d_train = xgb.DMatrix(x_tr_set1, label=y_tr_set1)
d_test = xgb.DMatrix(x_test_set1, label=y_test_set1)

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

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

xgdmat = xgb.DMatrix(x_tr_set1,y_tr_set1)
predict_y = bst.predict(d_test)
```

```
[0] train-logloss:0.66132 valid-logloss:0.66267
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10] train-logloss:0.512494 valid-logloss:0.522546
[20] train-logloss:0.458697 valid-logloss:0.479964
[30] train-logloss:0.431779 valid-logloss:0.462746
[40] train-logloss:0.414161 valid-logloss:0.455426
[50] train-logloss:0.398559 valid-logloss:0.45054
```

```
[60] train-logloss:0.382897 valid-logloss:0.446608
[70] train-logloss:0.366914 valid-logloss:0.443654
[80] train-logloss:0.355275 valid-logloss:0.441785
[90] train-logloss:0.344767 valid-logloss:0.440792
[100] train-logloss:0.332641 valid-logloss:0.440071
[110] train-logloss:0.321628 valid-logloss:0.439307
[120] train-logloss:0.311207 valid-logloss:0.437578
[130] train-logloss:0.298065 valid-logloss:0.436514
[140] train-logloss:0.289942 valid-logloss:0.436009
[150] train-logloss:0.280797 valid-logloss:0.435821
[160] train-logloss:0.272584 valid-logloss:0.435975
[170] train-logloss:0.264299 valid-logloss:0.434842
[180] train-logloss:0.255647 valid-logloss:0.434292
[190] train-logloss:0.248177 valid-logloss:0.434594
[200] train-logloss:0.241086 valid-logloss:0.434431
Stopping. Best iteration:
[183] train-logloss:0.252949 valid-logloss:0.43408
```

In [234]:

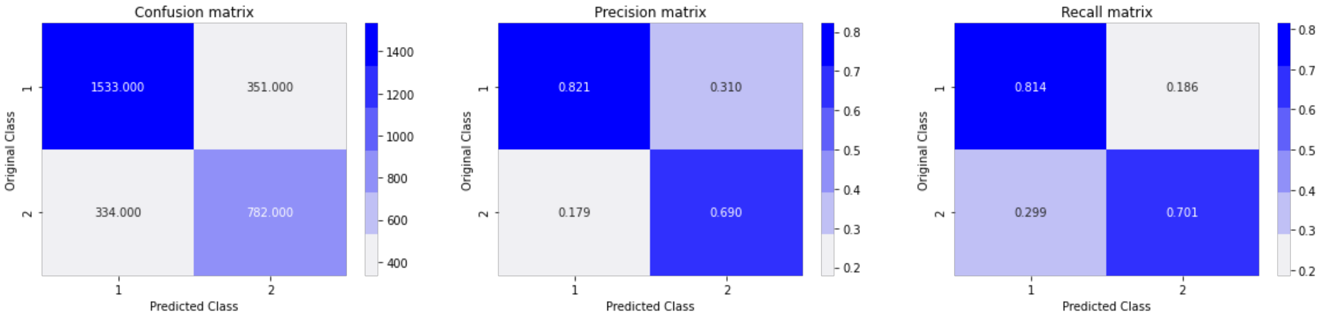
```
print("The test log loss is:",log_loss(y_test_set1, predict_y,  eps=1e-15))
```

```
The test log loss is: 0.43423951045549375
```

In [235]:

```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test_set1, predicted_y)
```

```
Total number of data points : 3000
```



In [236]:

```
predict_y = bst.predict(d_train)
loss_tr = log_loss(y_tr_set1, predict_y,  eps=1e-15)
predict_y = bst.predict(d_test)
loss_test = log_loss(y_test_set1, predict_y,  eps=1e-15)
```

In [237]:

```
summary.add_row(['TFIDF_w2v', 'XGBoost by GridSearchCV', 'learning_rate = '+str(best_lr)+' &
max_depth = '+str(best_max_depth) , loss_tr, loss_test ])
print(summary)
```

```
+------------+-----------------------------------------+--------------------------------------+-----
------------+---------------------+
| Vectorizer |                  Model                  |         Best Hyperparameters         |  Log
oss on Train  |   Log loss on Test  |
+------------+-----------------------------------------+--------------------------------------+-----
------------+---------------------+
| TFIDF_w2v  |            Random dumb model            |                  -                   |  0.8
9857413369713 |  0.8839580037167362 |
|   TFIDF    |            Random dumb model            |                  -                   |  0.8
1386222435589 |  0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search |             alpha = 0.1              |
0.5181022850363174 |  0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |            alpha = 0.001             |  0.5
|    TFIDF    | Logistic Regression by iterative search |            alpha = 0.001           |  0.5
04491052297945 |  0.5149092374073378 |
| TFIDF_w2v   |       Linear SVM by iterative search     |       L1 Norm & alpha = 0.0001      |  0.5
27068330818554  |  0.5528737395236669 |
| TFIDF_w2v   |       Linear SVM by iterative search     |        L2 Norm & alpha = 0.1        |  0.5
956886197710739 |  0.6080006713561955 |
|    TFIDF    |       Linear SVM by iterative search     |       L1 Norm & alpha = 1e-05       |
0.49838297495456246 |  0.5148041636287534 |
|    TFIDF    |       Linear SVM by iterative search     |       L2 Norm & alpha = 0.001       |
0.5389807644172705 |  0.551159201065387   |
| TFIDF_w2v   |          XGBoost by GridSearchCV          | learning_rate = 0.1 & max_depth = 4 | 0.23
898285190906504 | 0.43423951045549375 |
+------------+------------------------------------------+-------------------------------------+-----
------------+---------------------+
```

9.4.2 on set2 TFIDF

In [238]:

```
# https://www.kaggle.com/phunter/xgboost-with-gridsearchcv
xgb_clf = xgb.XGBClassifier()

parameters = {'objective' : ['binary:logistic'],  'eval_metric' : ['logloss'],
        'max_depth': [1,2,3,4],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
         }
d_train = xgb.DMatrix(x_tr_set2, label=y_tr_set2)
d_test = xgb.DMatrix(x_test_set2, label=y_test_set2)

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

In [239]:

```
clf = GridSearchCV(xgb_clf, parameters, n_jobs = -1,
                   cv=3,
                   scoring='neg_log_loss',
                   verbose=2, refit=True)
```

In [240]:

```
clf.fit(x_tr_set2, y_tr_set2)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:  1.8min finished
```

Out[240]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=-1,
             param_grid={'eval_metric': ['logloss'],
                         'learning_rate': [0.001, 0.01, 0.1, 0.2, 0, 3],
                         'max_depth': [1, 2, 3, 4],
                         'objective': ['binary:logistic']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_log_loss', verbose=2)
```

In [241]:

```
for param_name in sorted(clf.best_params_.keys()):
    print((param_name, clf.best_params_[param_name]))
```

```
('eval_metric', 'logloss')
('learning_rate', 0.2)
('max_depth', 4)
('objective', 'binary:logistic')
```

In [242]:

```
best_lr = clf.best_params_['learning_rate']
best_max_depth = clf.best_params_['max_depth']
```

In [243]:

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['learning_rate'] = best_lr
params['max_depth'] = best_max_depth

d_train = xgb.DMatrix(x_tr_set2, label=y_tr_set2)
d_test = xgb.DMatrix(x_test_set2, label=y_test_set2)

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

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

xgdmat = xgb.DMatrix(x_tr_set2,y_tr_set2)
predict_y = bst.predict(d_test)
```

```
[0] train-logloss:0.633313 valid-logloss:0.635351
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10] train-logloss:0.458116 valid-logloss:0.477098
[20] train-logloss:0.423672 valid-logloss:0.453158
[30] train-logloss:0.40006 valid-logloss:0.441863
[40] train-logloss:0.381075 valid-logloss:0.434494
[50] train-logloss:0.362261 valid-logloss:0.429763
[60] train-logloss:0.347543 valid-logloss:0.426033
[70] train-logloss:0.335782 valid-logloss:0.422541
[80] train-logloss:0.323992 valid-logloss:0.421173
[90] train-logloss:0.316225 valid-logloss:0.421302
[100] train-logloss:0.307727 valid-logloss:0.420663
[110] train-logloss:0.300978 valid-logloss:0.420736
[120] train-logloss:0.292785 valid-logloss:0.41968
[130] train-logloss:0.285708 valid-logloss:0.419546
[140] train-logloss:0.278457 valid-logloss:0.421014
[150] train-logloss:0.272805 valid-logloss:0.421247
Stopping. Best iteration:
[133] train-logloss:0.282118 valid-logloss:0.418883
```

In [244]:

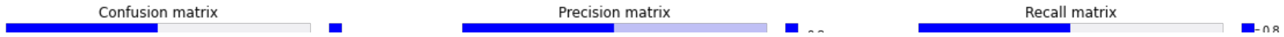
```
print("The test log loss is:",log_loss(y_test_set2, predict_y,  eps=1e-15))
```

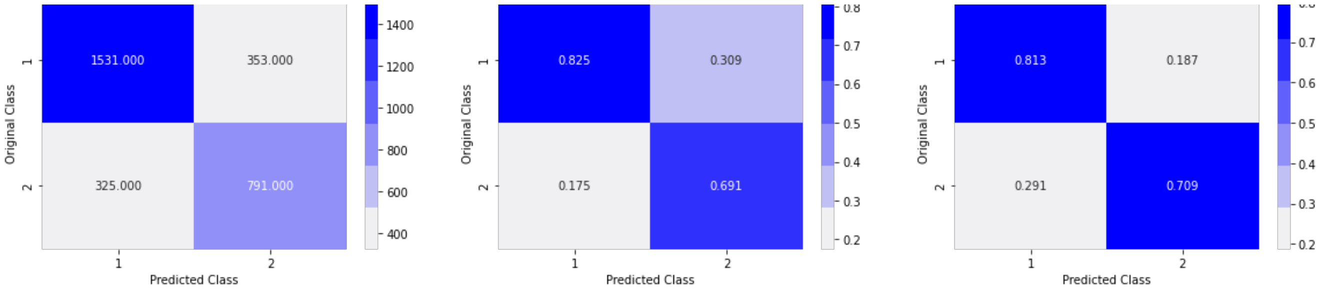
```
The test log loss is: 0.42154032061294855
```

In [245]:

```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test_set2, predicted_y)
```

```
Total number of data points : 3000
```





In [246]:

```
predict_y = bst.predict(d_train)
loss_tr = log_loss(y_tr_set2, predict_y,  eps=1e-15)
predict_y = bst.predict(d_test)
loss_test = log_loss(y_test_set2, predict_y,  eps=1e-15)
```

In [247]:

```
summary.add_row(['TFIDF', 'XGBoost by GridSearchCV', 'learning_rate = '+str(best_lr)+' & max_depth
= '+str(best_max_depth) , loss_tr, loss_test ])
print(summary)
```

```
+------------+------------------------------------------+--------------------------------------+-----
------------+---------------------+
| Vectorizer |                  Model                   |          Best Hyperparameters        |  Log
oss on Train  |   Log loss on Test  |
+------------+------------------------------------------+--------------------------------------+-----
------------+---------------------+
| TFIDF_w2v  |             Random dumb model            |                  -                   |  0.8
9857413369713 |  0.8839580037167362 |
|   TFIDF    |             Random dumb model            |                  -                   |  0.8
1386222435589 |  0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search  |             alpha = 0.1              |
0.5181022850363174 |  0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search  |            alpha = 0.001             |  0.5
04491052297945 |  0.5149092374073378 |
| TFIDF_w2v  |      Linear SVM by iterative search      |       L1 Norm & alpha = 0.0001       |  0.5
27068330818554  |  0.5528737395236669 |
| TFIDF_w2v  |      Linear SVM by iterative search      |        L2 Norm & alpha = 0.1         |  0.5
956886197710739 |  0.6080006713561955 |
|   TFIDF    |      Linear SVM by iterative search      |       L1 Norm & alpha = 1e-05        |
0.49838297495456246 |  0.5148041636287534 |
|   TFIDF    |      Linear SVM by iterative search      |       L2 Norm & alpha = 0.001        |
0.5389807644172705 |  0.551159201065387  |
| TFIDF_w2v  |         XGBoost by GridSearchCV          | learning_rate = 0.1 & max_depth = 4  | 0.23
898285190906504 | 0.43423951045549375 |
|   TFIDF    |         XGBoost by GridSearchCV          | learning_rate = 0.2 & max_depth = 4  | 0.27
033581958191283 | 0.42154032061294855 |
+------------+------------------------------------------+--------------------------------------+-----
------------+---------------------+
```

10.Conclusion

In [248]:

```
print(summary)
```

```
+------------+------------------------------------------+--------------------------------------+-----
------------+---------------------+
| Vectorizer |                  Model                   |          Best Hyperparameters        |  Log
oss on Train  |   Log loss on Test  |
+------------+------------------------------------------+--------------------------------------+-----
------------+---------------------+
| TFIDF_w2v  |             Random dumb model            |                  -                   |  0.8
9857413369713 |  0.8839580037167362 |
|   TFIDF    |             Random dumb model            |                  -                   |  0.8
1386222435589 |  0.9077642168454928 |
| TFIDF_w2v  | Logistic Regression by iterative search  |             alpha = 0.1              |
0.5181022850363174 |  0.5504720745790216 |
0.5181022850563174 |  0.5504720745790216 |
|   TFIDF    | Logistic Regression by iterative search |            alpha = 0.001             |  0.5
04491052297945 |  0.5149092374073378 |
| TFIDF_w2v  |      Linear SVM by iterative search     |       L1 Norm & alpha = 0.0001       |  0.5
27068330818554  |  0.5528737395236669 |
| TFIDF_w2v  |      Linear SVM by iterative search     |        L2 Norm & alpha = 0.1         |  0.5
956886197710739 |  0.6080006713561955 |
|   TFIDF    |      Linear SVM by iterative search     |       L1 Norm & alpha = 1e-05        |
0.49838297495456246 |  0.5148041636287534 |
|   TFIDF    |      Linear SVM by iterative search     |       L2 Norm & alpha = 0.001        |
0.5389807644172705 |  0.551159201065387  |
| TFIDF_w2v  |         XGBoost by GridSearchCV         | learning_rate = 0.1 & max_depth = 4 | 0.23
898285190906504 | 0.43423951045549375 |
|   TFIDF    |         XGBoost by GridSearchCV         | learning_rate = 0.2 & max_depth = 4 | 0.27
033581958191283 | 0.42154032061294855 |
+------------+-----------------------------------------+--------------------------------------+-----
------------+---------------------+
```

11.Observations and in detail Summary

1. From EDA of the data we observed that:
   - The dataset is imbalanced with 63-37% approx to non duplicate to duplicate question pairs
   - There is very less percentage of Unique question being repeated.
   - There are no repeated questions pairs in Dataset.
   - There are no NULL entries.
2. Basic Feature Engineering
   - We created 11 Features before preprocessing the data and observed the following:
     - 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)
     - The distributions of the word_Common feature in similar and non-similar questions are highly overlapping
3. Preprocessing of the data
   - Removing html tags
   - Removing Punctuations
   - Performing stemming
   - Removing Stopwords
   - Expanding contractions etc.
4. Advance Feature Engineering
   - We created 15 Advance Features after preprocessing the data and observed the following:
     - There are few words which occurs frequently in both the cases when the questions are similar and not similiar.
     - From the pair plots of few features, It can be observed that there are few non duplcate pairs of questions which are linearly spearable, althought there is a lot of overlapping of distribution as well.
     - The TSNE plot very well proves the above point that there are few clusters of non duplicate pairs which are not overlapping at all with clusters of duplicate pairs.
     - The TSNE plot also shows that there are majority of clusters of non duplicate pairs which are overlapping with clusters of duplicate pairs.
5. Splitting of Data
   - Data is splitted into 70-30 ratio into Train and Test
6. Featurizing Text Data
   - Created TFIDF for Train and Test data by fitting vectorizer on train data and transforming it on train and test, also limiting the max_fetures to 100.
   - created TFIDF weighted Word to Vec for Train and Test data appropriately.
7. Creating Data ready for Training Models
   - Set1: comprises of all 11 basic features + 15 advance features + 96 features of TFIDF weighted word to vec of Que 1 + 96 features of TFIDF weighted word to vec of Que 2.
   - Set2: comprises of all 11 basic features + 15 advance features + 100 features of TFIDF of Que 1 + 100 features of TFIDF of Que 2.
8. Training Models
   - Random Dumb Model gives log loss of 0.89 and 0.90 on Set1 and Set2 respectively.
   - Logistic Regression works better on Set2 as compared to Set1 Data.
   - Linear SVM works better on Set2 as compared to Set1 Data.
   - XGBoost works marginally better on Set2 as compared to Set1 Data.