Quora Question Pairs

1. Business Problem

1.1 Description

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

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

```
Credits: Kaggle
```

Problem Statement:

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

1.2 Sources/Useful Links

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

Useful Links

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

1. "id","qid1","qid2","question1","question2","is_duplicate"
2. "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"
3. "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"
4. "7","15","16","How can I be a good geologist?","What should I do to be a great geologist?","1"
5. "11","23","24","How do I read and find my YouTube comments?","How can I see all my Youtube comments?","1"

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc
import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
```

3.1 Reading data and basic stats

In [2]:

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

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

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

In [3]:

```
df.head()
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [4]:

```
df_sample= df.sample(n = 20000)
```

In [5]:

```
df = df_sample
```

In [6]:

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

Out[6]:

	id	qid1	qid2	question1	question2	is_duplicate
271918	271918	390102	208688	How do I relieve lower back pain and stiffness?	What is the best way to relieve lower back pain?	1
284108	284108	317372	13720	Why do some educated people in developed count...	Why do some people with PhD's join groups like...	1

In [7]:

```
y_true = df["is_duplicate"]
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20000 entries, 271918 to 132810
Data columns (total 6 columns):
id              20000 non-null int64
qid1            20000 non-null int64
qid2            20000 non-null int64
question1       20000 non-null object
question2       19999 non-null object
is_duplicate    20000 non-null int64
dtypes: int64(4), object(2)
memory usage: 1.1+ MB
```

We are given a minimal number of data fields here, consisting of:

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

3.2.1 Distribution of data points among output classes

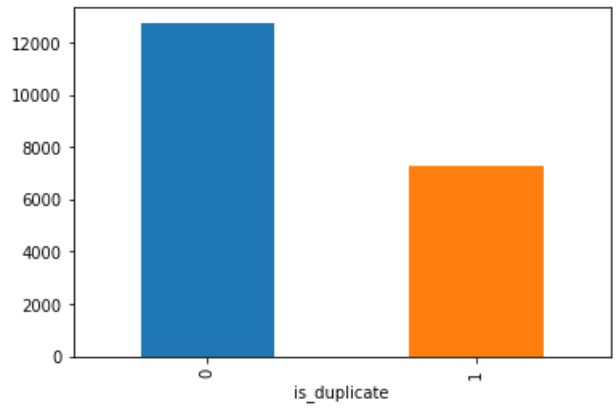
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [9]:

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

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2cc8a5be470>
```



In [10]:

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

```
~> Total number of question pairs for training:
   20000
```

In [11]:

```
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):
   63.61%

~> Question pairs are Similar (is_duplicate = 1):
   36.39%
```

3.2.2 Number of unique questions

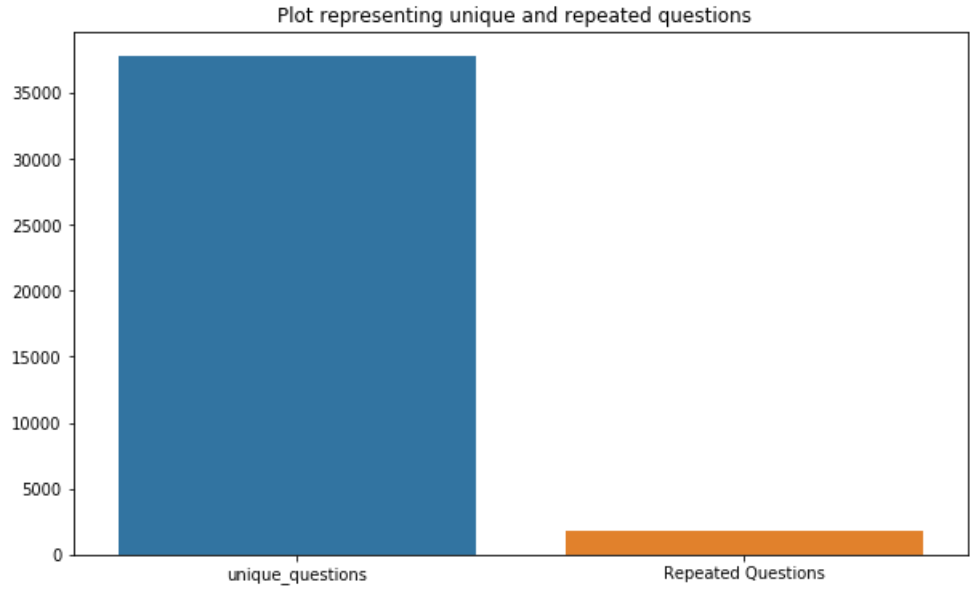
In [12]:

```
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: 37751

Number of unique questions that appear more than one time: 1821 (4.823713279118434%)

Max number of times a single question is repeated: 11
```

In [13]:

```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

plt.figure(figsize=(10, 6))
plt.title ("Plot representing unique and repeated questions  ")
sns.barplot(x,y)
plt.show()
```



3.2.3 Checking for Duplicates

In [14]:

```
#checking whether there are any repeated pair of questions

pair_duplicates =
df[['qid1','qid2','is_duplicate']].groupby(['qid1','qid2']).count().reset_index()

print ("Number of duplicate questions",(pair_duplicates).shape[0] - df.shape[0])
```

```
Number of duplicate questions 0
```

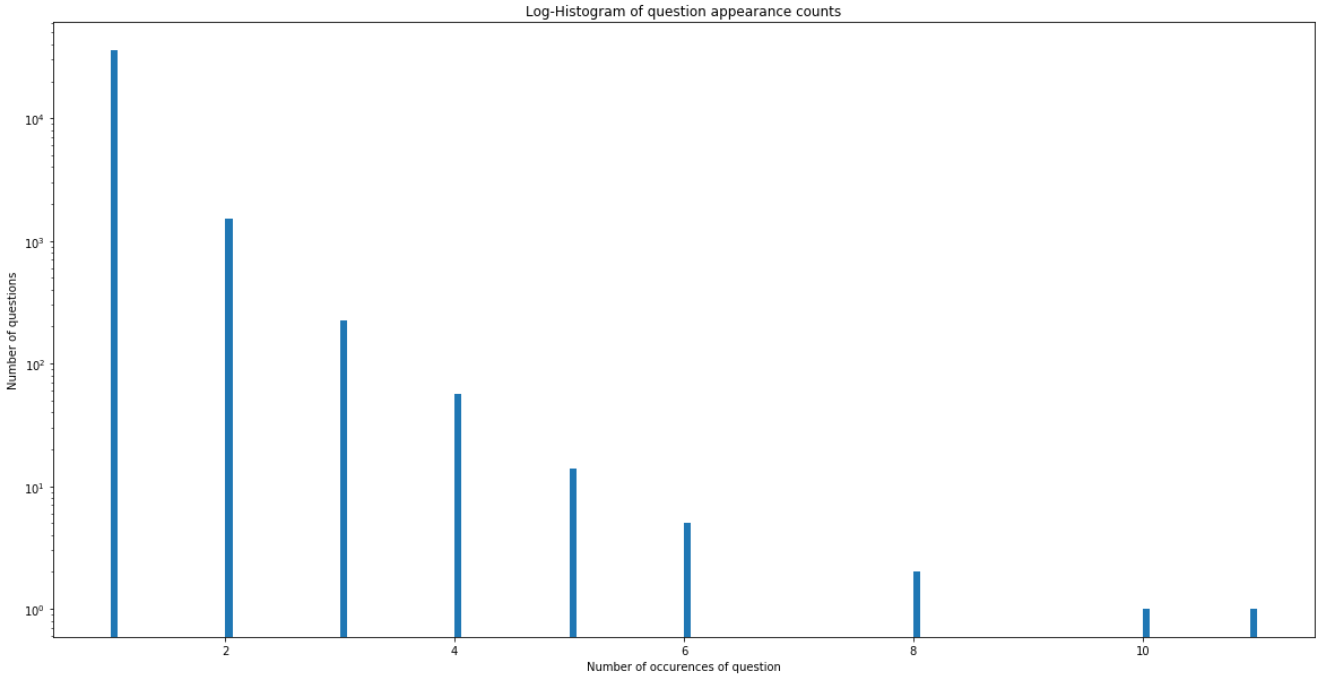
3.2.4 Number of occurrences of each question

In [15]:

```
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: 11
```



3.2.5 Checking for NULL values

In [16]:

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

```
            id    qid1    qid2                         question1 question2  \
201841  201841  303951  174364  How can I create an Android app?       NaN

        is_duplicate
201841             0
```

In [17]:

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

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- freq_qid1 = Frequency of qid1's
- freq_qid2 = Frequency of qid2's
- q1len = Length of q1
- q2len = Length of q2
- q1_n_words = Number of words in Question 1
- q2_n_words = Number of words in Question 2
- word_Common = (Number of common unique words in Question 1 and Question 2)

- word_Total =(Total num of words in Question 1 + Total num of words in Question 2)
- word_share = (word_common)/(word_Total)
- freq_q1+freq_q2 = sum total of frequency of qid1 and qid2
- freq_q1-freq_q2 = absolute difference of frequency of qid1 and qid2

In [18]:

```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

    def normalized_word_Common(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * len(w1 & w2)
    df['word_Common'] = df.apply(normalized_word_Common, axis=1)

    def normalized_word_Total(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * (len(w1) + len(w2))
    df['word_Total'] = df.apply(normalized_word_Total, axis=1)

    def normalized_word_share(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * len(w1 & w2)/(len(w1) + len(w2))
    df['word_share'] = df.apply(normalized_word_share, axis=1)

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

    df.to_csv("df_fe_without_preprocessing_train.csv", index=False)

df.head()
```

Out[18]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_wor
0	68765	16044	118786	How can I have more casual sex in India?	How can I have casual sex with a girl in India?	1	1	1	40	47	9	11
1	173815	268046	268047	How much money does a TV weather person earn?	How much money does a mid level (appears on TV...	0	1	1	45	108	9	21
2	329526	456195	456196	What or who has influenced your life the most ...	Which person has influenced your life the most?	1	1	1	54	47	11	8
3	315567	56798	194687	What's the difference between the title "found...	What is the difference between the term founde...	1	1	1	67	63	9	10
				Is it okay to	At what							

4	400916	370954	534222	be physical in relationship fo...	stage in a relationship does an Indian...	0	1	1	13	12	13	10	

3.3.1 Analysis of some of the extracted features

In [19]:

```
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])
```

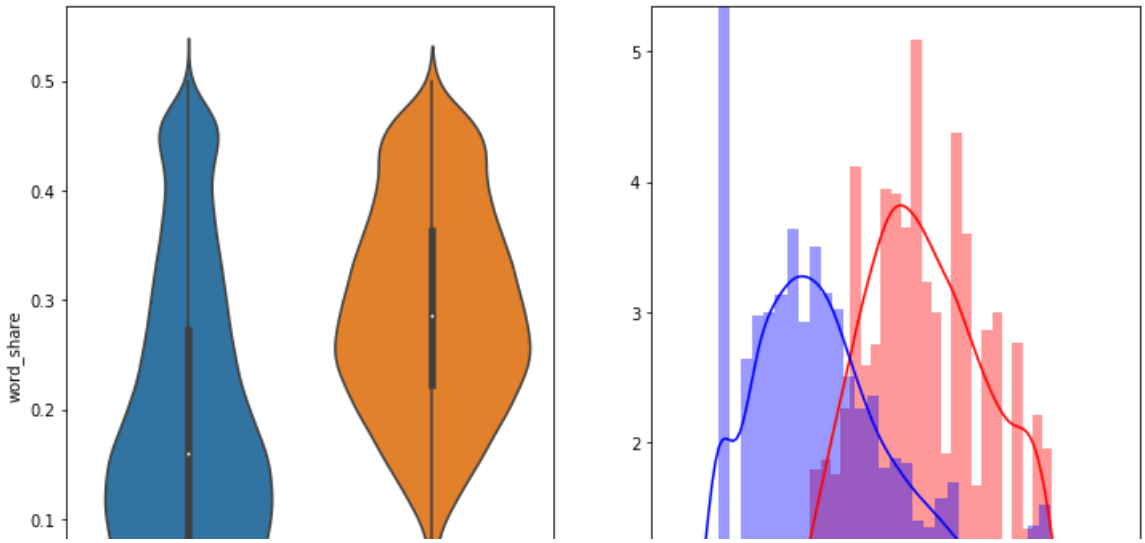
```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 4
Number of Questions with minimum length [question2] : 1
```

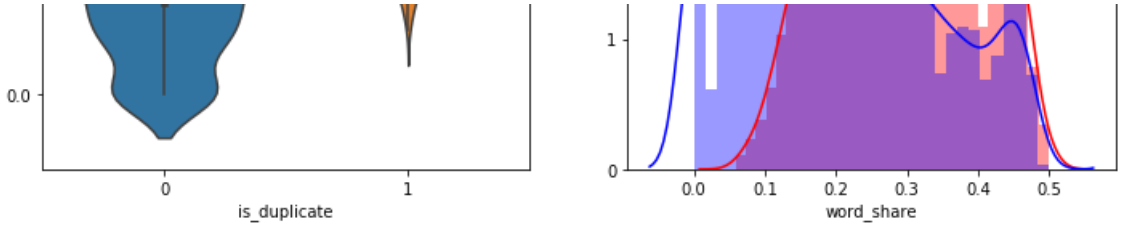
3.3.1.1 Feature: word_share

In [20]:

```
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()
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` inst
ead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`,
which will result either in an error or a different result.

C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.

C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```



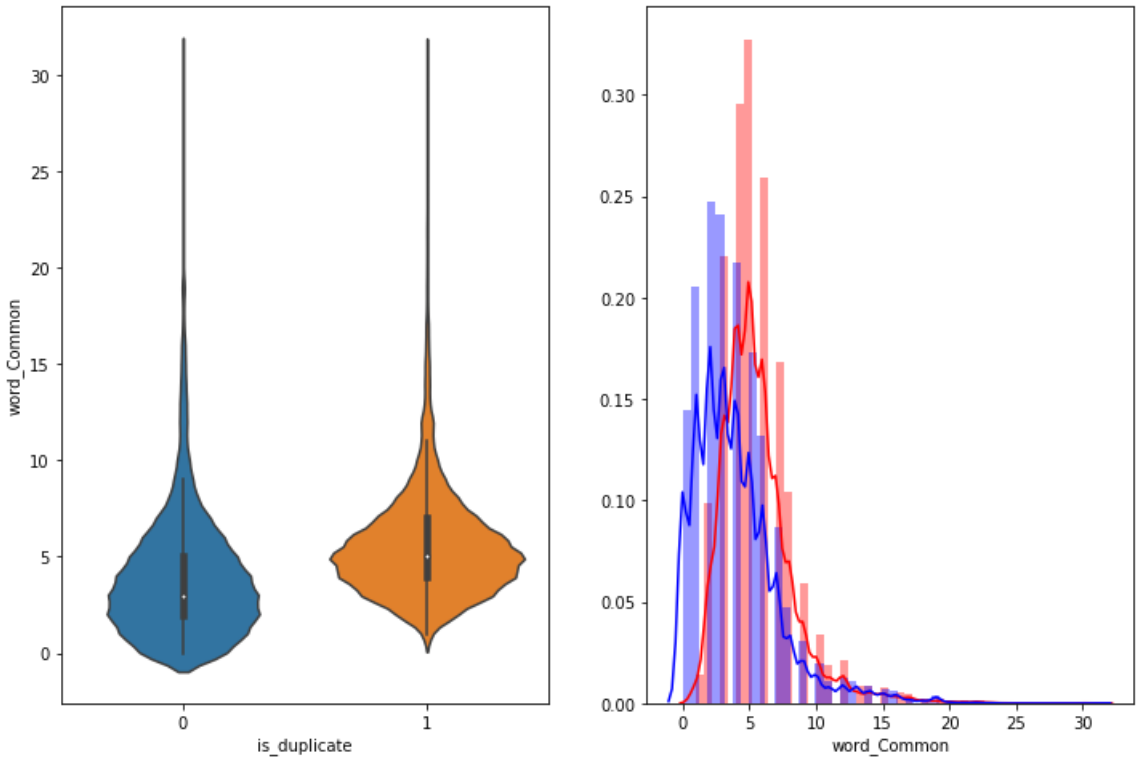


3.3.1.2 Feature: word_Common

In [21]:

```
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()
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` inst
ead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`,
which will result either in an error or a different result.

C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.

C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```



1.2.1 : EDA: Advanced Feature Extraction

In [22]:

```
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 [23]:

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

Out[23]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
0	68765	16044	118786	How can I have more casual sex in India?	How can I have casual sex with a girl in India?	1	1	1	40	47	9	11
1	173815	268046	268047	How much money does a TV weather person earn?	How much money does a mid level (appears on TV...	0	1	1	45	108	9	21

3.4 Preprocessing of Text

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

In [24]:

```
# 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'
", "can not")\
                           .replace("n't", " not").replace("what's", "what is").replace("it's", "it
is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").replace("'s", " own
)\
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar
")\
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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


    return x
```

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

- Token: You get a token by splitting sentence a space
- Stop_Word : stop words as per NLTK.
- Word : A token that is not a stop_word Features:
- cwc_min : Ratio of common_word_count to min lenghth of word count of Q1 and Q2
- cwc_min = common_word_count / (min(len(q1_words), len(q2_words))
- cwc_max : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
- cwc_max = common_word_count / (max(len(q1_words), len(q2_words))
- csc_min : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2
- csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))
- csc_max : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
- csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))
- ctc_min : Ratio of common_token_count to min lenghth of token count of Q1 and Q2
- ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))

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

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

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

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

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

- fuzz_ratio : https://github.com/seatgeek/fuzzywuzzy#usage 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 [25]:

```
def get_token_features(q1, q2):
    token_features = [0.0]*10

    # Converting the Sentence into Tokens:
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])

    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])

    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))

    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))

    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)

    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

    # First word of both question is same or not
    token_features[7] = int(q1_tokens[0] == q2_tokens[0])

    token_features[8] = abs(len(q1_tokens) - len(q2_tokens))

    #Average Token Length of both Questions
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

def extract_features(df):
    # preprocessing each question
    df["question1"] = df["question1"].fillna("").apply(preprocess)
    df["question2"] = df["question2"].fillna("").apply(preprocess)
```

```
    print("token features...")

    # Merging Features with dataset

    token_features = df.apply(lambda x: get_token_features(x["question1"], x["question2"]), axis=1)

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

    #Computing Fuzzy Features and Merging with Dataset

    # do read this blog: http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
    # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-function-to-compare-2-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 [26]:

```
import fuzzywuzzy as fuzzy
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = df_sample
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[26]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_w
0	68765	16044	118786	how can i have more casual sex in india	how can i have casual sex with a girl in india	1	0.999967	0.749981	0.833319	0.714276	...	0.727266	1
1	173815	268046	268047	how much money does a tv weather person earn	how much money does a mid level appears on tv...	0	0.666656	0.307690	0.999967	0.499992	...	0.333332	1

2 rows × 21 columns

In [27]:

```
y_true = df['is_duplicate']
```

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

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

In [28]:

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

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

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

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

```
Number of data points in class 1 (duplicate pairs) : 14932
Number of data points in class 0 (non duplicate pairs) : 25068
```

In [29]:

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

textp_w = open(path.join(d, 'train_p.txt'),encoding="utf8").read()
textn_w = open(path.join(d, 'train_n.txt'),encoding="utf8").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))
```

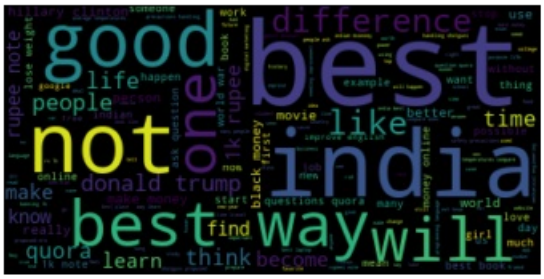
```
Total number of words in duplicate pair questions : 815775
Total number of words in non duplicate pair questions : 1623200
```

Word Clouds generated from duplicate pair question's text

In [30]:

```
wc = WordCloud(background_color="black", 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
```

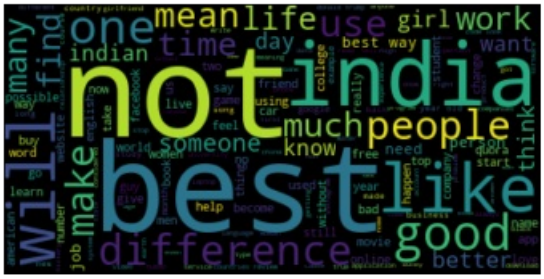


Word Clouds generated from non duplicate pair question's text

In [31]:

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

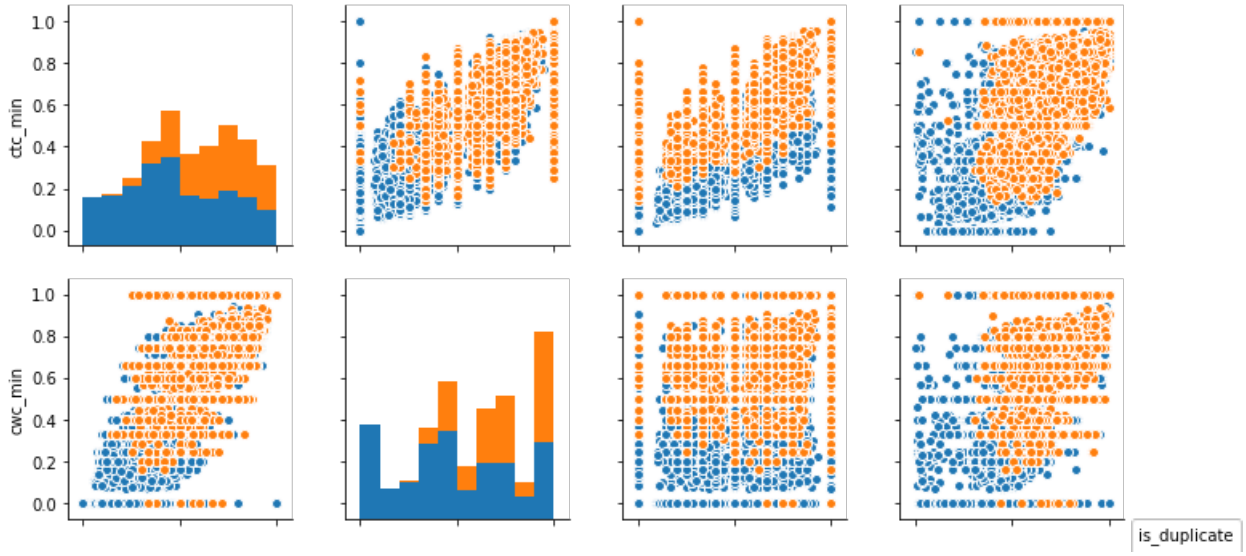
```
Word Cloud for non-Duplicate Question pairs:
```

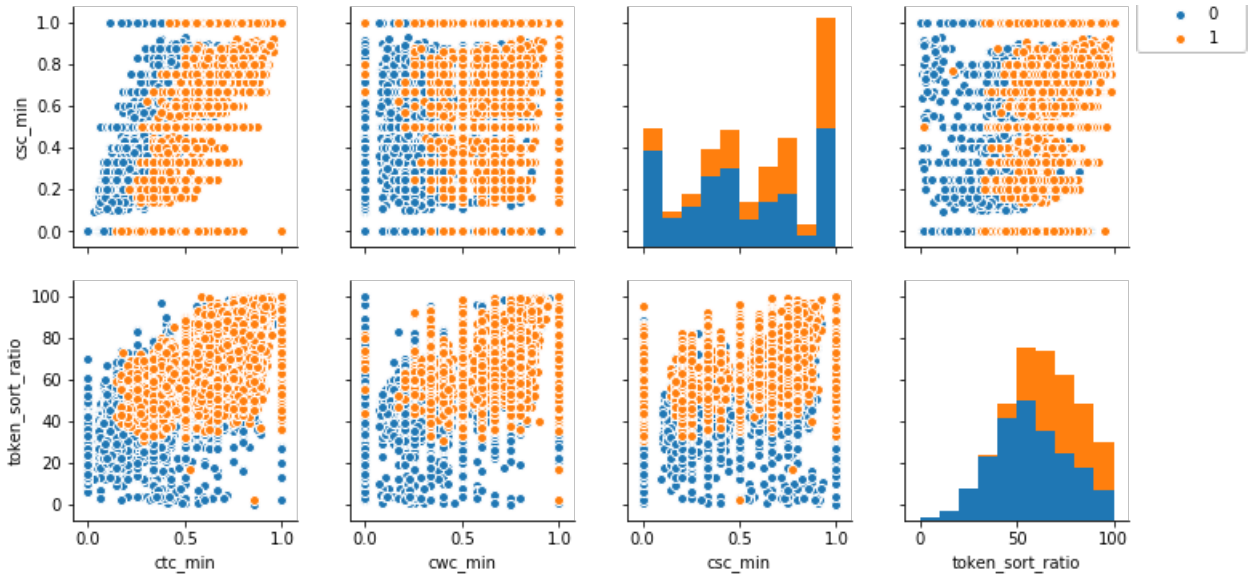


3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

In [32]:

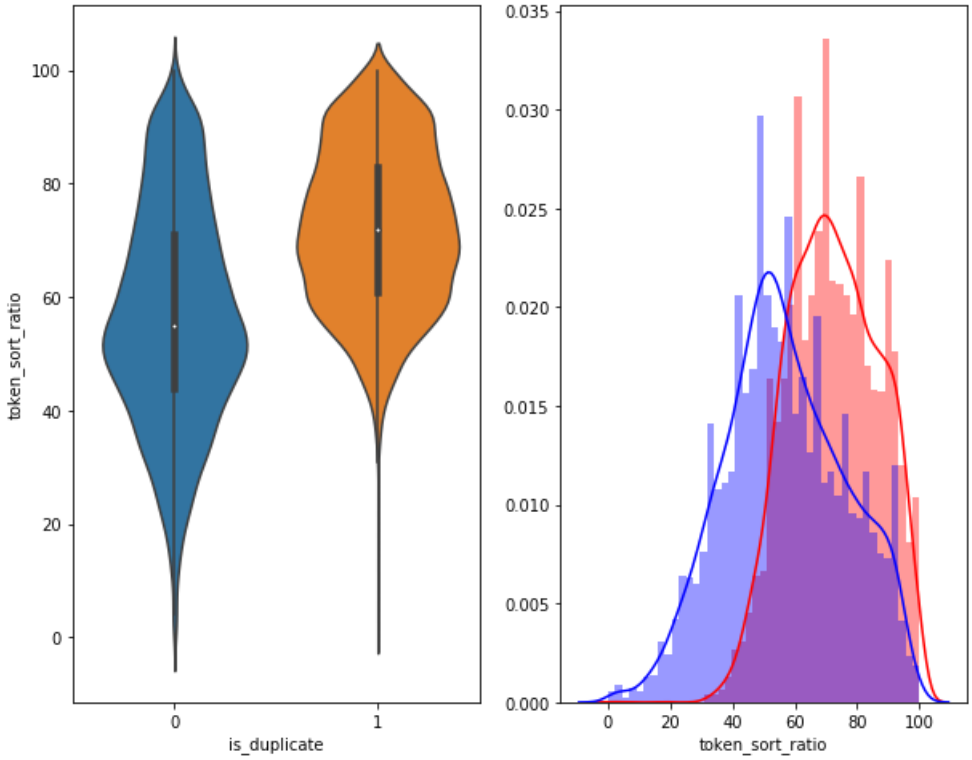
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```





In [33]:

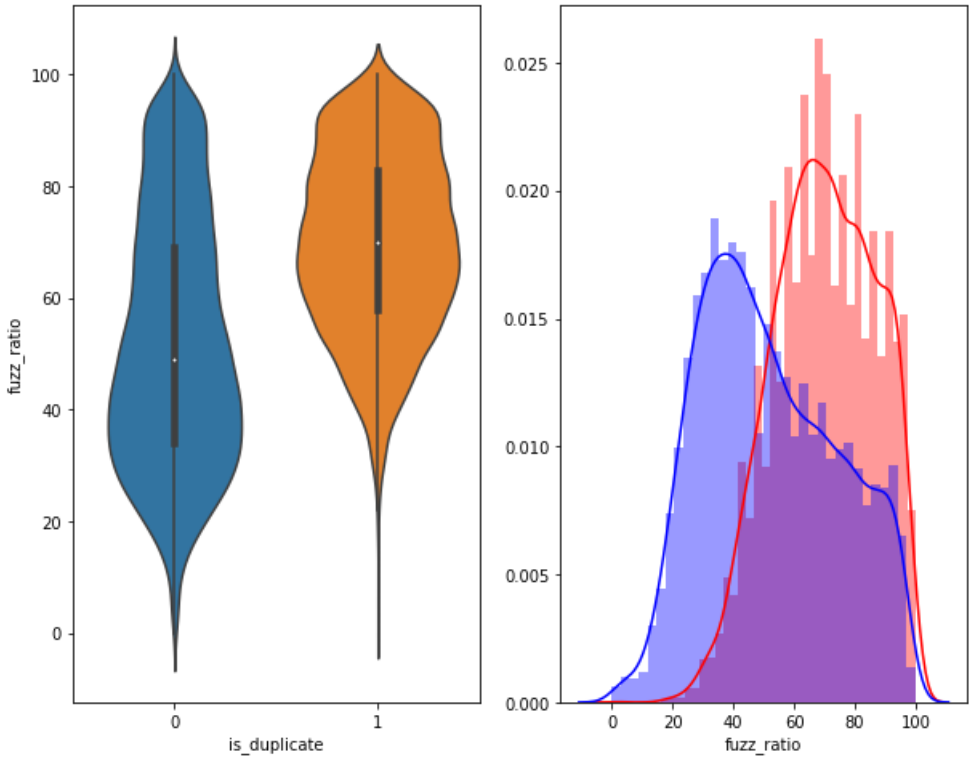
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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 [34]:

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

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

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

```
plt.show()
```



3.5.2 Visualization

In [35]:

```
# 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 [36]:

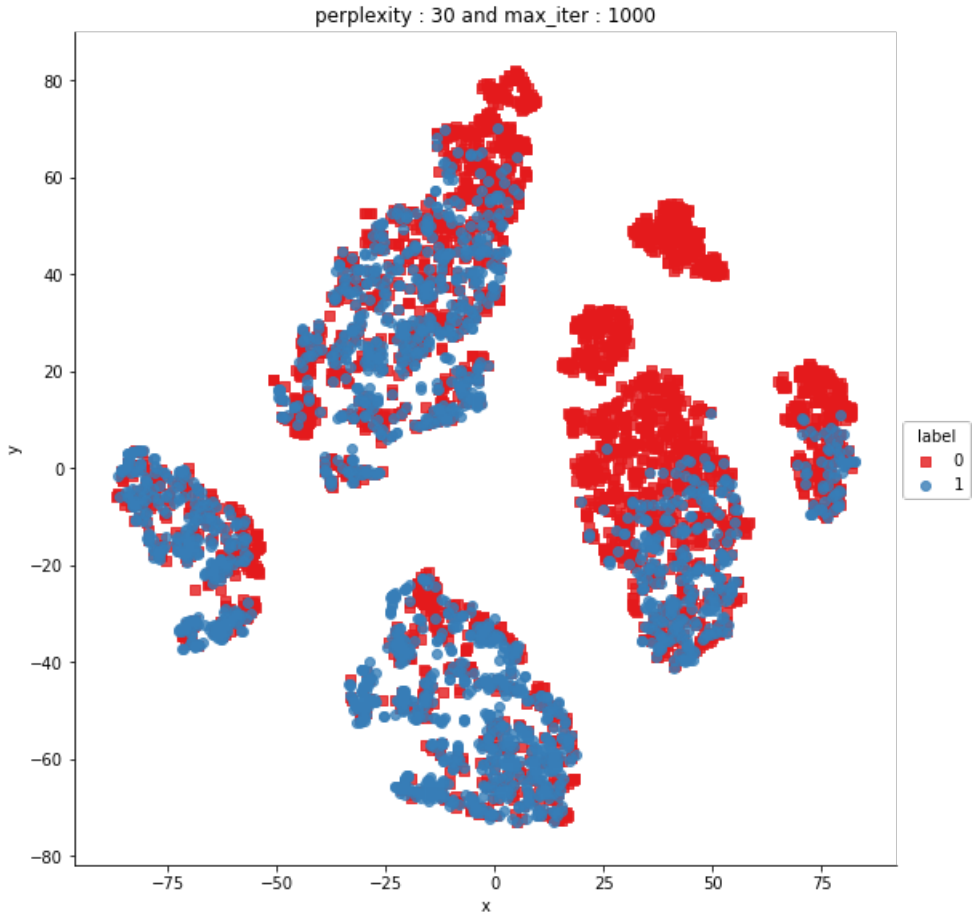
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.046s...
[t-SNE] Computed neighbors for 5000 samples in 1.061s...
[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: 0.143517
[t-SNE] Computed conditional probabilities in 0.624s
[t-SNE] Iteration 50: error = 82.7032547, gradient norm = 0.0480684 (50 iterations in 8.724s)
[t-SNE] Iteration 100: error = 70.8729019, gradient norm = 0.0087711 (50 iterations in 6.010s)
[t-SNE] Iteration 150: error = 69.1524887, gradient norm = 0.0055995 (50 iterations in 5.302s)
[t-SNE] Iteration 200: error = 68.4579544, gradient norm = 0.0036117 (50 iterations in 5.735s)
```

```
[t-SNE] Iteration 250: error = 68.0476608, gradient norm = 0.0027974 (50 iterations in 5.814s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 68.047661
[t-SNE] Iteration 300: error = 1.8392988, gradient norm = 0.0012084 (50 iterations in 5.298s)
[t-SNE] Iteration 350: error = 1.4433486, gradient norm = 0.0004864 (50 iterations in 5.836s)
[t-SNE] Iteration 400: error = 1.2777306, gradient norm = 0.0002794 (50 iterations in 6.951s)
[t-SNE] Iteration 450: error = 1.1881936, gradient norm = 0.0001884 (50 iterations in 6.358s)
[t-SNE] Iteration 500: error = 1.1333711, gradient norm = 0.0001402 (50 iterations in 5.558s)
[t-SNE] Iteration 550: error = 1.0981762, gradient norm = 0.0001146 (50 iterations in 5.662s)
[t-SNE] Iteration 600: error = 1.0749811, gradient norm = 0.0001008 (50 iterations in 5.521s)
[t-SNE] Iteration 650: error = 1.0592834, gradient norm = 0.0000927 (50 iterations in 5.787s)
[t-SNE] Iteration 700: error = 1.0482447, gradient norm = 0.0000835 (50 iterations in 6.911s)
[t-SNE] Iteration 750: error = 1.0397959, gradient norm = 0.0000788 (50 iterations in 6.741s)
[t-SNE] Iteration 800: error = 1.0334688, gradient norm = 0.0000715 (50 iterations in 5.495s)
[t-SNE] Iteration 850: error = 1.0275393, gradient norm = 0.0000681 (50 iterations in 5.113s)
[t-SNE] Iteration 900: error = 1.0225691, gradient norm = 0.0000654 (50 iterations in 5.915s)
[t-SNE] Iteration 950: error = 1.0184410, gradient norm = 0.0000601 (50 iterations in 5.332s)
[t-SNE] Iteration 1000: error = 1.0146582, gradient norm = 0.0000574 (50 iterations in 5.884s)
[t-SNE] KL divergence after 1000 iterations: 1.014658
```

In [37]:

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

# draw the plot in appropriate place in the grid
sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",markers=['s','o
'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



In [38]:

```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
```

```
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.024s...
[t-SNE] Computed neighbors for 5000 samples in 0.910s...
[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: 0.143517
[t-SNE] Computed conditional probabilities in 0.503s
[t-SNE] Iteration 50: error = 83.0664520, gradient norm = 0.0388372 (50 iterations in 25.856s)
[t-SNE] Iteration 100: error = 69.8177338, gradient norm = 0.0030177 (50 iterations in 13.641s)
[t-SNE] Iteration 150: error = 68.5871811, gradient norm = 0.0016061 (50 iterations in 11.169s)
[t-SNE] Iteration 200: error = 68.0660400, gradient norm = 0.0012006 (50 iterations in 11.401s)
[t-SNE] Iteration 250: error = 67.7567368, gradient norm = 0.0008811 (50 iterations in 11.639s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.756737
[t-SNE] Iteration 300: error = 1.5812273, gradient norm = 0.0007478 (50 iterations in 15.111s)
[t-SNE] Iteration 350: error = 1.2287776, gradient norm = 0.0002089 (50 iterations in 19.892s)
[t-SNE] Iteration 400: error = 1.0784249, gradient norm = 0.0001058 (50 iterations in 19.730s)
[t-SNE] Iteration 450: error = 1.0034419, gradient norm = 0.0000614 (50 iterations in 19.836s)
[t-SNE] Iteration 500: error = 0.9628308, gradient norm = 0.0000480 (50 iterations in 19.430s)
[t-SNE] Iteration 550: error = 0.9401002, gradient norm = 0.0000425 (50 iterations in 19.223s)
[t-SNE] Iteration 600: error = 0.9266233, gradient norm = 0.0000386 (50 iterations in 19.202s)
[t-SNE] Iteration 650: error = 0.9175710, gradient norm = 0.0000363 (50 iterations in 19.147s)
[t-SNE] Iteration 700: error = 0.9114534, gradient norm = 0.0000315 (50 iterations in 19.791s)
[t-SNE] Iteration 750: error = 0.9062611, gradient norm = 0.0000282 (50 iterations in 19.240s)
[t-SNE] Iteration 800: error = 0.9013082, gradient norm = 0.0000285 (50 iterations in 20.480s)
[t-SNE] Iteration 850: error = 0.8969287, gradient norm = 0.0000278 (50 iterations in 19.721s)
[t-SNE] Iteration 900: error = 0.8929207, gradient norm = 0.0000261 (50 iterations in 19.409s)
[t-SNE] Iteration 950: error = 0.8897073, gradient norm = 0.0000257 (50 iterations in 19.636s)
[t-SNE] Iteration 1000: error = 0.8867765, gradient norm = 0.0000251 (50 iterations in 19.446s)
[t-SNE] KL divergence after 1000 iterations: 0.886777
```

In [39]:

```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

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

3.6 Featurizing text data with tfidf

In [40]:

```
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 [41]:

```
# avoid decoding problems
df = pd.read_csv("train.csv")

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
# ----------------- python 3 ---------------------
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [42]:

```
df.shape
```

Out[42]:

```
(404290, 6)
```

In [43]:

```
df = df.sample( n = 20000)
```

In [44]:

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

Out[44]:

	id	qid1	qid2	question1	question2	is_duplicate
351337	351337	480175	480176	How is popcorn made?	How are Popcorners made?	1
270606	270606	81304	388531	What are some ways of removing permanent marke...	How do you remove permanent marker from leather?	0

In [45]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
tfidf = TfidfVectorizer(lowercase=False,max_features=1000 )
tfidf_vec_q1 = tfidf.fit_transform(df['question1'])
tfidf_vec_q2 = tfidf.fit_transform(df['question2'])
```

In [46]:

```
# Assigning the Sparse Matrices of Q1 and Q2 Sparse Matrices
q1_vec = tfidf_vec_q1[:df.shape[0]]
q2_vec = tfidf_vec_q2[:df.shape[0]]
```

In [47]:

```
q1_vec = q1_vec.toarray()
q2_vec = q2_vec.toarray()
```

In [48]:

```
q1_vec.shape
```

Out[48]:

```
(20000, 1000)
```

In [49]:

```
q2_vec.shape
```

Out[49]:

```
(20000, 1000)
```

In [50]:

```
index_x = ["x_" + str(i) for i in range(0, q1_vec.shape[1]) ]
index_y = ["y_" + str(i) for i in range(0, q2_vec.shape[1])]
df3_q1 = pd.DataFrame(data=q1_vec, columns=index_x)
df3_q2 = pd.DataFrame(data=q2_vec, columns=index_y)
```

In [51]:

```
df3_q2.head(5)
```

Out[51]:

	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	...	y_990	y_991	y_992	y_993	y_994	y_995	y_996	y_997	y_99
0	0.0	0.0	0.0	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	0.000000	0.00000
1	0.0	0.0	0.0	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	0.336794	0.00000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.292347	0.0	0.0	0.0	0.0	0.0	0.0	0.207821	0.00000
3	0.0	0.0	0.0	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	0.000000	0.00000
4	0.0	0.0	0.0	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	0.000000	0.26435

5 rows × 1000 columns

In [52]:

```
#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 [53]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [54]:

```
df1.head(2)
```

Out[54]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_
0	68765	1	0.999967	0.749981	0.833319	0.714276	0.888879	0.727266	1	1	2
1	173815	0	0.666656	0.307690	0.999967	0.499992	0.777769	0.333332	1	1	12

In [55]:

```
df2.head(2)
```

Out[55]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+
0	68765	1	1	40	47	9	11	8.0	20.0	0.40	2
1	173815	1	1	45	108	9	21	7.0	28.0	0.25	2

In [56]:

```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape[1])
print("Number of features in question2 w2v  dataframe :", df3_q2.shape[1])
```

```
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df3_q1.shape[1]+df3_q2.
shape[1])
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 1000
Number of features in question2 w2v  dataframe : 1000
Number of features in final dataframe  : 2029
```

In [57]:

```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1['id']=df1['id']
    df3_q2['id']=df1['id']
    df1  = df1.merge(df2, on='id',how='left')
    df2  = df3_q1.merge(df3_q2, on='id',how='left')
    result  = df1.merge(df2, on='id',how='left')
    result.to_csv('final_features.csv')
```

4. Machine Learning Models

In [58]:

```
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
```

4.1 Reading data from file and storing into sql table

In [59]:

```
data = pd.read_csv("final_features.csv")
```

In [60]:

```
data.shape
```

Out[60]:

```
(20000, 2029)
```

In [61]:

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

Out[61]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	y_990	y_
0	0	68765	1	0.999967	0.749981	0.833319	0.714276	0.888879	0.727266	1	...	0.0	0.(
1	1	173815	0	0.666656	0.307690	0.999967	0.499992	0.777769	0.333332	1	...	0.0	0.(

2 rows × 2029 columns

In [62]:

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

In [63]:

```
data.head()
```

Out[63]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	...	
1	0.666656	0.307690	0.999967	0.499992	0.777769	0.333332	1	1	12	15.0	...	0.0
2	0.999950	0.666644	0.799984	0.444440	0.749991	0.545450	0	0	3	9.5	...	0.0
3	0.749981	0.749981	0.999980	0.999980	0.727266	0.727266	1	1	0	11.0	...	0.3
4	0.799984	0.499994	0.499994	0.444440	0.615380	0.470585	0	0	4	15.0	...	0.0
5	0.749981	0.749981	0.999975	0.799984	0.777769	0.777769	1	1	0	9.0	...	0.0

5 rows × 2026 columns

4.2 Converting strings to numerics

In [64]:

```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
cols = list(data.columns)
for i in cols:
    data[i] = data[i].apply(pd.to_numeric)
    print(i)
```

```
cwc_min
```

```
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
x_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_100
x_101
x_102
x_103
x_104
x_105
x_106
x_107
x_108
x_109
x_110
x_111
x_112
x_113
x_114
x_115
x_116
x_117
x_118
x_119
x_120
x_121
x_122
x_123
x_124
x_125
x_126
x_127
x_128

x_129
x_130
x_131
x_132
x_133
x_134
x_135
x_136
x_137
x_138
x_139
x_140
x_141
x_142
x_143
x_144
x_145
x_146
x_147
x_148
x_149
x_150
x_151
x_152
x_153
x_154
x_155
x_156
x_157
x_158
x_159
x_160
x_161
x_162
x_163
x_164
x_165
x_166
x_167
x_168
x_169
x_170
x_171
x_172
x_173
x_174
x_175
x_176
x_177
x_178
x_179
x_180
x_181
x_182
x_183
x_184
x_185
x_186
x_187
x_188
x_189
x_190
x_191
x_192
x_193
x_194
x_195
x_196
x_197
x_198
x_199
x_200
x_201
x_202
x_203
x_204
x_205

x_206
x_207
x_208
x_209
x_210
x_211
x_212
x_213
x_214
x_215
x_216
x_217
x_218
x_219
x_220
x_221
x_222
x_223
x_224
x_225
x_226
x_227
x_228
x_229
x_230
x_231
x_232
x_233
x_234
x_235
x_236
x_237
x_238
x_239
x_240
x_241
x_242
x_243
x_244
x_245
x_246
x_247
x_248
x_249
x_250
x_251
x_252
x_253
x_254
x_255
x_256
x_257
x_258
x_259
x_260
x_261
x_262
x_263
x_264
x_265
x_266
x_267
x_268
x_269
x_270
x_271
x_272
x_273
x_274
x_275
x_276
x_277
x_278
x_279
x_280
x_281
x_282

x_283
x_284
x_285
x_286
x_287
x_288
x_289
x_290
x_291
x_292
x_293
x_294
x_295
x_296
x_297
x_298
x_299
x_300
x_301
x_302
x_303
x_304
x_305
x_306
x_307
x_308
x_309
x_310
x_311
x_312
x_313
x_314
x_315
x_316
x_317
x_318
x_319
x_320
x_321
x_322
x_323
x_324
x_325
x_326
x_327
x_328
x_329
x_330
x_331
x_332
x_333
x_334
x_335
x_336
x_337
x_338
x_339
x_340
x_341
x_342
x_343
x_344
x_345
x_346
x_347
x_348
x_349
x_350
x_351
x_352
x_353
x_354
x_355
x_356
x_357
x_358
x_359

x_360
x_361
x_362
x_363
x_364
x_365
x_366
x_367
x_368
x_369
x_370
x_371
x_372
x_373
x_374
x_375
x_376
x_377
x_378
x_379
x_380
x_381
x_382
x_383
x_384
x_385
x_386
x_387
x_388
x_389
x_390
x_391
x_392
x_393
x_394
x_395
x_396
x_397
x_398
x_399
x_400
x_401
x_402
x_403
x_404
x_405
x_406
x_407
x_408
x_409
x_410
x_411
x_412
x_413
x_414
x_415
x_416
x_417
x_418
x_419
x_420
x_421
x_422
x_423
x_424
x_425
x_426
x_427
x_428
x_429
x_430
x_431
x_432
x_433
x_434
x_435
x_436

x_437
x_438
x_439
x_440
x_441
x_442
x_443
x_444
x_445
x_446
x_447
x_448
x_449
x_450
x_451
x_452
x_453
x_454
x_455
x_456
x_457
x_458
x_459
x_460
x_461
x_462
x_463
x_464
x_465
x_466
x_467
x_468
x_469
x_470
x_471
x_472
x_473
x_474
x_475
x_476
x_477
x_478
x_479
x_480
x_481
x_482
x_483
x_484
x_485
x_486
x_487
x_488
x_489
x_490
x_491
x_492
x_493
x_494
x_495
x_496
x_497
x_498
x_499
x_500
x_501
x_502
x_503
x_504
x_505
x_506
x_507
x_508
x_509
x_510
x_511
x_512
x_513

x_513
x_514
x_515
x_516
x_517
x_518
x_519
x_520
x_521
x_522
x_523
x_524
x_525
x_526
x_527
x_528
x_529
x_530
x_531
x_532
x_533
x_534
x_535
x_536
x_537
x_538
x_539
x_540
x_541
x_542
x_543
x_544
x_545
x_546
x_547
x_548
x_549
x_550
x_551
x_552
x_553
x_554
x_555
x_556
x_557
x_558
x_559
x_560
x_561
x_562
x_563
x_564
x_565
x_566
x_567
x_568
x_569
x_570
x_571
x_572
x_573
x_574
x_575
x_576
x_577
x_578
x_579
x_580
x_581
x_582
x_583
x_584
x_585
x_586
x_587
x_588
x_589
x_590

x_590
x_591
x_592
x_593
x_594
x_595
x_596
x_597
x_598
x_599
x_600
x_601
x_602
x_603
x_604
x_605
x_606
x_607
x_608
x_609
x_610
x_611
x_612
x_613
x_614
x_615
x_616
x_617
x_618
x_619
x_620
x_621
x_622
x_623
x_624
x_625
x_626
x_627
x_628
x_629
x_630
x_631
x_632
x_633
x_634
x_635
x_636
x_637
x_638
x_639
x_640
x_641
x_642
x_643
x_644
x_645
x_646
x_647
x_648
x_649
x_650
x_651
x_652
x_653
x_654
x_655
x_656
x_657
x_658
x_659
x_660
x_661
x_662
x_663
x_664
x_665
x_666
x_667

x_667
x_668
x_669
x_670
x_671
x_672
x_673
x_674
x_675
x_676
x_677
x_678
x_679
x_680
x_681
x_682
x_683
x_684
x_685
x_686
x_687
x_688
x_689
x_690
x_691
x_692
x_693
x_694
x_695
x_696
x_697
x_698
x_699
x_700
x_701
x_702
x_703
x_704
x_705
x_706
x_707
x_708
x_709
x_710
x_711
x_712
x_713
x_714
x_715
x_716
x_717
x_718
x_719
x_720
x_721
x_722
x_723
x_724
x_725
x_726
x_727
x_728
x_729
x_730
x_731
x_732
x_733
x_734
x_735
x_736
x_737
x_738
x_739
x_740
x_741
x_742
x_743
x_744

x_744
x_745
x_746
x_747
x_748
x_749
x_750
x_751
x_752
x_753
x_754
x_755
x_756
x_757
x_758
x_759
x_760
x_761
x_762
x_763
x_764
x_765
x_766
x_767
x_768
x_769
x_770
x_771
x_772
x_773
x_774
x_775
x_776
x_777
x_778
x_779
x_780
x_781
x_782
x_783
x_784
x_785
x_786
x_787
x_788
x_789
x_790
x_791
x_792
x_793
x_794
x_795
x_796
x_797
x_798
x_799
x_800
x_801
x_802
x_803
x_804
x_805
x_806
x_807
x_808
x_809
x_810
x_811
x_812
x_813
x_814
x_815
x_816
x_817
x_818
x_819
x_820

x_821
x_822
x_823
x_824
x_825
x_826
x_827
x_828
x_829
x_830
x_831
x_832
x_833
x_834
x_835
x_836
x_837
x_838
x_839
x_840
x_841
x_842
x_843
x_844
x_845
x_846
x_847
x_848
x_849
x_850
x_851
x_852
x_853
x_854
x_855
x_856
x_857
x_858
x_859
x_860
x_861
x_862
x_863
x_864
x_865
x_866
x_867
x_868
x_869
x_870
x_871
x_872
x_873
x_874
x_875
x_876
x_877
x_878
x_879
x_880
x_881
x_882
x_883
x_884
x_885
x_886
x_887
x_888
x_889
x_890
x_891
x_892
x_893
x_894
x_895
x_896
x_897

x_898
x_899
x_900
x_901
x_902
x_903
x_904
x_905
x_906
x_907
x_908
x_909
x_910
x_911
x_912
x_913
x_914
x_915
x_916
x_917
x_918
x_919
x_920
x_921
x_922
x_923
x_924
x_925
x_926
x_927
x_928
x_929
x_930
x_931
x_932
x_933
x_934
x_935
x_936
x_937
x_938
x_939
x_940
x_941
x_942
x_943
x_944
x_945
x_946
x_947
x_948
x_949
x_950
x_951
x_952
x_953
x_954
x_955
x_956
x_957
x_958
x_959
x_960
x_961
x_962
x_963
x_964
x_965
x_966
x_967
x_968
x_969
x_970
x_971
x_972
x_973
x_974

x_975
x_976
x_977
x_978
x_979
x_980
x_981
x_982
x_983
x_984
x_985
x_986
x_987
x_988
x_989
x_990
x_991
x_992
x_993
x_994
x_995
x_996
x_997
x_998
x_999
y_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_100
y_101
y_102
y_103
y_104
y_105
y_106
y_107
y_108
y_109
y_110
y_111
y_112
y_113
y_114
y_115
y_116
y_117
y_118
y_119
y_120
y_121
y_122
y_123
y_124
y_125
y_126
y_127
y_128

y_129
y_130
y_131
y_132
y_133
y_134
y_135
y_136
y_137
y_138
y_139
y_140
y_141
y_142
y_143
y_144
y_145
y_146
y_147
y_148
y_149
y_150
y_151
y_152
y_153
y_154
y_155
y_156
y_157
y_158
y_159
y_160
y_161
y_162
y_163
y_164
y_165
y_166
y_167
y_168
y_169
y_170
y_171
y_172
y_173
y_174
y_175
y_176
y_177
y_178
y_179
y_180
y_181
y_182
y_183
y_184
y_185
y_186
y_187
y_188
y_189
y_190
y_191
y_192
y_193
y_194
y_195
y_196
y_197
y_198
y_199
y_200
y_201
y_202
y_203
y_204
y_205

y_206
y_207
y_208
y_209
y_210
y_211
y_212
y_213
y_214
y_215
y_216
y_217
y_218
y_219
y_220
y_221
y_222
y_223
y_224
y_225
y_226
y_227
y_228
y_229
y_230
y_231
y_232
y_233
y_234
y_235
y_236
y_237
y_238
y_239
y_240
y_241
y_242
y_243
y_244
y_245
y_246
y_247
y_248
y_249
y_250
y_251
y_252
y_253
y_254
y_255
y_256
y_257
y_258
y_259
y_260
y_261
y_262
y_263
y_264
y_265
y_266
y_267
y_268
y_269
y_270
y_271
y_272
y_273
y_274
y_275
y_276
y_277
y_278
y_279
y_280
y_281
y_282

y_283
y_284
y_285
y_286
y_287
y_288
y_289
y_290
y_291
y_292
y_293
y_294
y_295
y_296
y_297
y_298
y_299
y_300
y_301
y_302
y_303
y_304
y_305
y_306
y_307
y_308
y_309
y_310
y_311
y_312
y_313
y_314
y_315
y_316
y_317
y_318
y_319
y_320
y_321
y_322
y_323
y_324
y_325
y_326
y_327
y_328
y_329
y_330
y_331
y_332
y_333
y_334
y_335
y_336
y_337
y_338
y_339
y_340
y_341
y_342
y_343
y_344
y_345
y_346
y_347
y_348
y_349
y_350
y_351
y_352
y_353
y_354
y_355
y_356
y_357
y_358
y_359

y_360
y_361
y_362
y_363
y_364
y_365
y_366
y_367
y_368
y_369
y_370
y_371
y_372
y_373
y_374
y_375
y_376
y_377
y_378
y_379
y_380
y_381
y_382
y_383
y_384
y_385
y_386
y_387
y_388
y_389
y_390
y_391
y_392
y_393
y_394
y_395
y_396
y_397
y_398
y_399
y_400
y_401
y_402
y_403
y_404
y_405
y_406
y_407
y_408
y_409
y_410
y_411
y_412
y_413
y_414
y_415
y_416
y_417
y_418
y_419
y_420
y_421
y_422
y_423
y_424
y_425
y_426
y_427
y_428
y_429
y_430
y_431
y_432
y_433
y_434
y_435
y_436

y_437
y_438
y_439
y_440
y_441
y_442
y_443
y_444
y_445
y_446
y_447
y_448
y_449
y_450
y_451
y_452
y_453
y_454
y_455
y_456
y_457
y_458
y_459
y_460
y_461
y_462
y_463
y_464
y_465
y_466
y_467
y_468
y_469
y_470
y_471
y_472
y_473
y_474
y_475
y_476
y_477
y_478
y_479
y_480
y_481
y_482
y_483
y_484
y_485
y_486
y_487
y_488
y_489
y_490
y_491
y_492
y_493
y_494
y_495
y_496
y_497
y_498
y_499
y_500
y_501
y_502
y_503
y_504
y_505
y_506
y_507
y_508
y_509
y_510
y_511
y_512
y_513

y_514
y_515
y_516
y_517
y_518
y_519
y_520
y_521
y_522
y_523
y_524
y_525
y_526
y_527
y_528
y_529
y_530
y_531
y_532
y_533
y_534
y_535
y_536
y_537
y_538
y_539
y_540
y_541
y_542
y_543
y_544
y_545
y_546
y_547
y_548
y_549
y_550
y_551
y_552
y_553
y_554
y_555
y_556
y_557
y_558
y_559
y_560
y_561
y_562
y_563
y_564
y_565
y_566
y_567
y_568
y_569
y_570
y_571
y_572
y_573
y_574
y_575
y_576
y_577
y_578
y_579
y_580
y_581
y_582
y_583
y_584
y_585
y_586
y_587
y_588
y_589
y_590

y_591
y_592
y_593
y_594
y_595
y_596
y_597
y_598
y_599
y_600
y_601
y_602
y_603
y_604
y_605
y_606
y_607
y_608
y_609
y_610
y_611
y_612
y_613
y_614
y_615
y_616
y_617
y_618
y_619
y_620
y_621
y_622
y_623
y_624
y_625
y_626
y_627
y_628
y_629
y_630
y_631
y_632
y_633
y_634
y_635
y_636
y_637
y_638
y_639
y_640
y_641
y_642
y_643
y_644
y_645
y_646
y_647
y_648
y_649
y_650
y_651
y_652
y_653
y_654
y_655
y_656
y_657
y_658
y_659
y_660
y_661
y_662
y_663
y_664
y_665
y_666
y_667

y_668
y_669
y_670
y_671
y_672
y_673
y_674
y_675
y_676
y_677
y_678
y_679
y_680
y_681
y_682
y_683
y_684
y_685
y_686
y_687
y_688
y_689
y_690
y_691
y_692
y_693
y_694
y_695
y_696
y_697
y_698
y_699
y_700
y_701
y_702
y_703
y_704
y_705
y_706
y_707
y_708
y_709
y_710
y_711
y_712
y_713
y_714
y_715
y_716
y_717
y_718
y_719
y_720
y_721
y_722
y_723
y_724
y_725
y_726
y_727
y_728
y_729
y_730
y_731
y_732
y_733
y_734
y_735
y_736
y_737
y_738
y_739
y_740
y_741
y_742
y_743
y_744

y_744
y_745
y_746
y_747
y_748
y_749
y_750
y_751
y_752
y_753
y_754
y_755
y_756
y_757
y_758
y_759
y_760
y_761
y_762
y_763
y_764
y_765
y_766
y_767
y_768
y_769
y_770
y_771
y_772
y_773
y_774
y_775
y_776
y_777
y_778
y_779
y_780
y_781
y_782
y_783
y_784
y_785
y_786
y_787
y_788
y_789
y_790
y_791
y_792
y_793
y_794
y_795
y_796
y_797
y_798
y_799
y_800
y_801
y_802
y_803
y_804
y_805
y_806
y_807
y_808
y_809
y_810
y_811
y_812
y_813
y_814
y_815
y_816
y_817
y_818
y_819
y_820
y_821

y_821
y_822
y_823
y_824
y_825
y_826
y_827
y_828
y_829
y_830
y_831
y_832
y_833
y_834
y_835
y_836
y_837
y_838
y_839
y_840
y_841
y_842
y_843
y_844
y_845
y_846
y_847
y_848
y_849
y_850
y_851
y_852
y_853
y_854
y_855
y_856
y_857
y_858
y_859
y_860
y_861
y_862
y_863
y_864
y_865
y_866
y_867
y_868
y_869
y_870
y_871
y_872
y_873
y_874
y_875
y_876
y_877
y_878
y_879
y_880
y_881
y_882
y_883
y_884
y_885
y_886
y_887
y_888
y_889
y_890
y_891
y_892
y_893
y_894
y_895
y_896
y_897
y_898

y_898
y_899
y_900
y_901
y_902
y_903
y_904
y_905
y_906
y_907
y_908
y_909
y_910
y_911
y_912
y_913
y_914
y_915
y_916
y_917
y_918
y_919
y_920
y_921
y_922
y_923
y_924
y_925
y_926
y_927
y_928
y_929
y_930
y_931
y_932
y_933
y_934
y_935
y_936
y_937
y_938
y_939
y_940
y_941
y_942
y_943
y_944
y_945
y_946
y_947
y_948
y_949
y_950
y_951
y_952
y_953
y_954
y_955
y_956
y_957
y_958
y_959
y_960
y_961
y_962
y_963
y_964
y_965
y_966
y_967
y_968
y_969
y_970
y_971
y_972
y_973
y_974
y_975

```
y_975
y_976
y_977
y_978
y_979
y_980
y_981
y_982
y_983
y_984
y_985
y_986
y_987
y_988
y_989
y_990
y_991
y_992
y_993
y_994
y_995
y_996
y_997
y_998
y_999
```

In [65]:

```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
y_true = list(map(int, y_true.values))
```

4.3 Random train test split(70:30)

In [66]:

```
X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

In [67]:

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

```
Number of data points in train data : (13999, 2026)
Number of data points in test data : (6000, 2026)
```

In [68]:

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

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6267590542181585 Class 1:  0.3732409457818416
---------- Distribution of output variable in train data ----------
Class 0:  0.37333333333333335 Class 1:  0.37333333333333335
```

In [69]:

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

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column
```

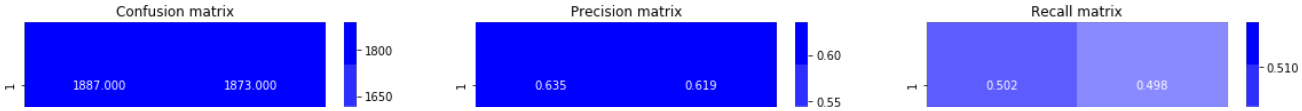
```
    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                              [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #      [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

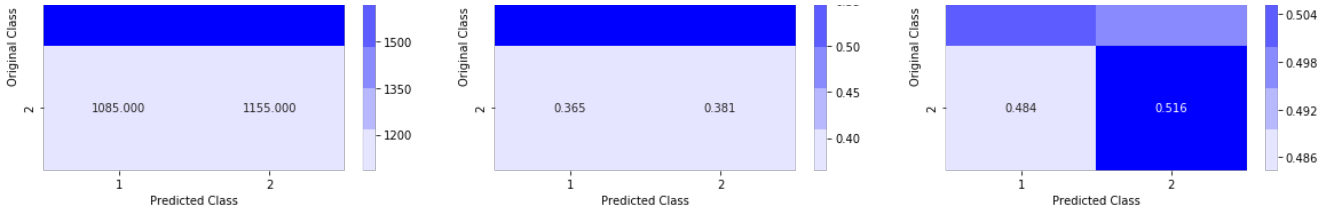
4.4 Building a random model (Finding worst-case log-loss)

In [70]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-15))

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

```
Log loss on Test Data using Random Model 0.8700255736979046
```





In [93]:

```
results=pd.DataFrame(columns=['Model', 'Featuraization', 'Hyperparameter', 'Train_loss','Test-
loss', ])
```

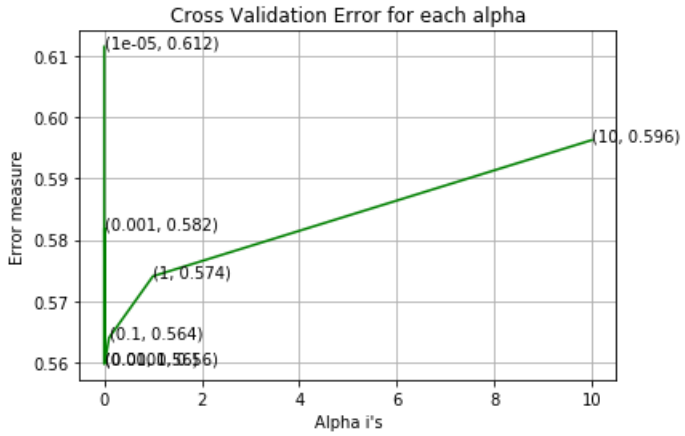
4.4 Logistic Regression with hyperparameter tuning

In [71]:

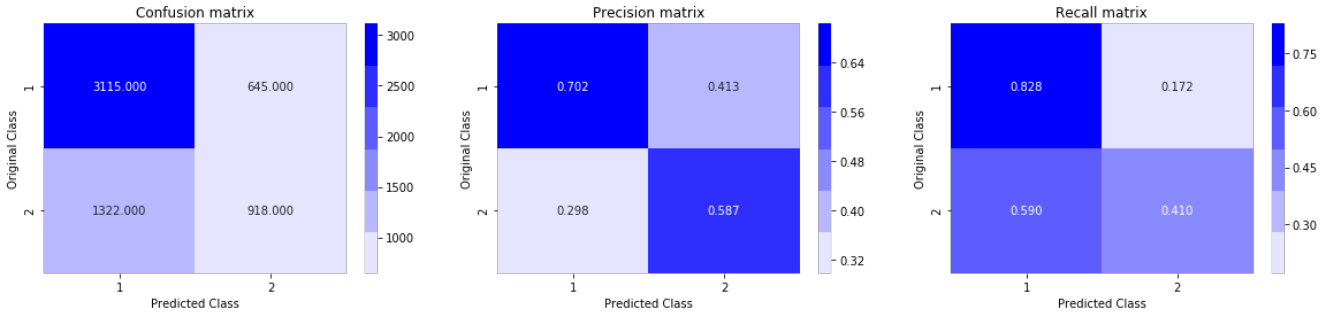
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
```

```
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6115483961310546
For values of alpha =  0.0001 The log loss is: 0.5597209010695454
For values of alpha =  0.001 The log loss is: 0.5818704081186798
For values of alpha =  0.01 The log loss is: 0.5597998364467709
For values of alpha =  0.1 The log loss is: 0.5639873739066641
For values of alpha =  1 The log loss is: 0.5740423038443414
For values of alpha =  10 The log loss is: 0.5962905057201728
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5599676132394837
For values of best alpha =  0.0001 The test log loss is: 0.5597209010695454
Total number of data points : 6000
```



In [94]:

```
new = ["Logistic-Regession",'TFIDF','alpha = 0.0001','0.559','0.559']
results.loc[0] = new
```

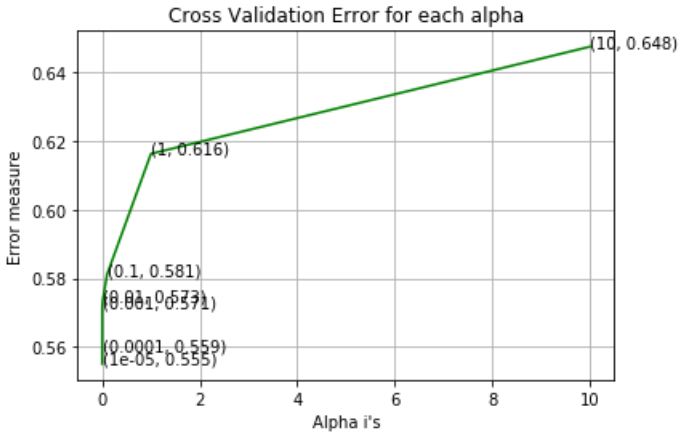
4.5 Linear SVM with hyperparameter tuning

In [72]:

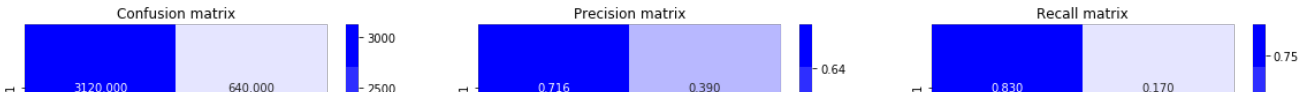
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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.

#-------------------------------
```

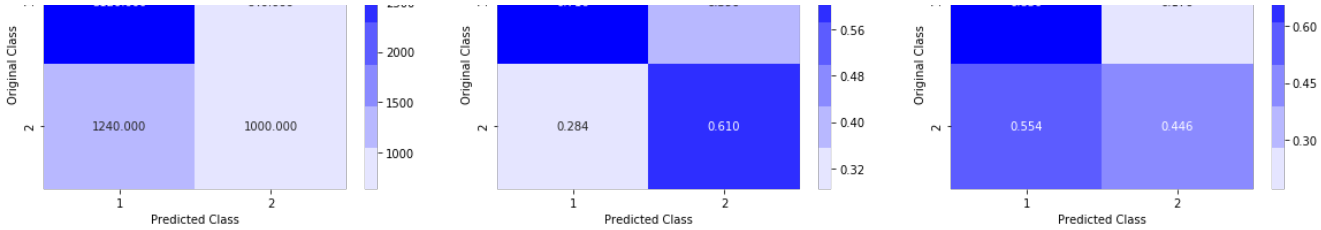
```
# video link:
#------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.5550291567960447
For values of alpha =  0.0001 The log loss is: 0.5589397519756102
For values of alpha =  0.001 The log loss is: 0.5714505127117191
For values of alpha =  0.01 The log loss is: 0.5733729872461801
For values of alpha =  0.1 The log loss is: 0.5811054457248479
For values of alpha =  1 The log loss is: 0.6162829404196747
For values of alpha =  10 The log loss is: 0.6475012095443965
```




```
For values of best alpha =  1e-05 The train log loss is: 0.5551932301332952
For values of best alpha =  1e-05 The test log loss is: 0.5550291567960447
Total number of data points : 6000
```





In [95]:

```
new = ['Linear SVM','TFIDF','alpha = 10e-05','0.555','0.555']
results.loc[1] = new
```

4.6 XGBoost with hyperparameter tuning

In [87]:

```
# Finding optimal number of base learners using k-fold CV ->
base_learn = [x for x in range(30, 120,10)]
base_learn
```

Out[87]:

```
[30, 40, 50, 60, 70, 80, 90, 100, 110]
```

In [88]:

```
# Learning rate values ->
#learning_rate = [x/10 for x in range(1,11)]
#learning_rate
```

In [89]:

```
# Max-depth values- >
#max_depth = [x for x in range(1,6)]
#max_depth
```

In [90]:

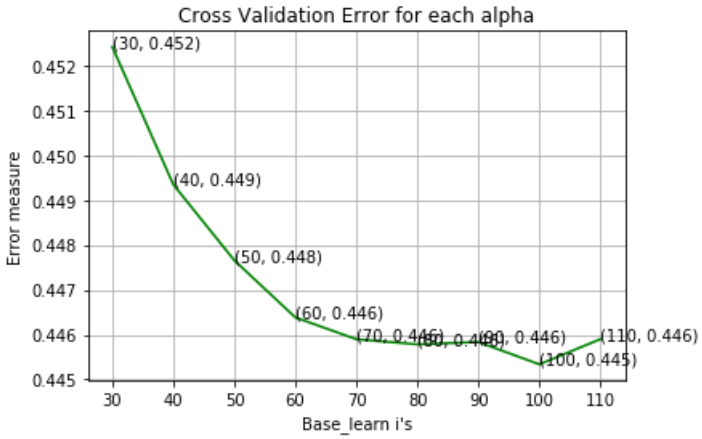
```
# Using RandomSearchCv to get optimal parameters ->
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
#params = {'n_estimators': base_learn, 'learning_rate': learning_rate, 'max_depth': max_depth}
#clf =xgb.XGBClassifier()
#model = GridSearchCV(clf, params)
#model.fit(X_train, y_train)
```

In [91]:

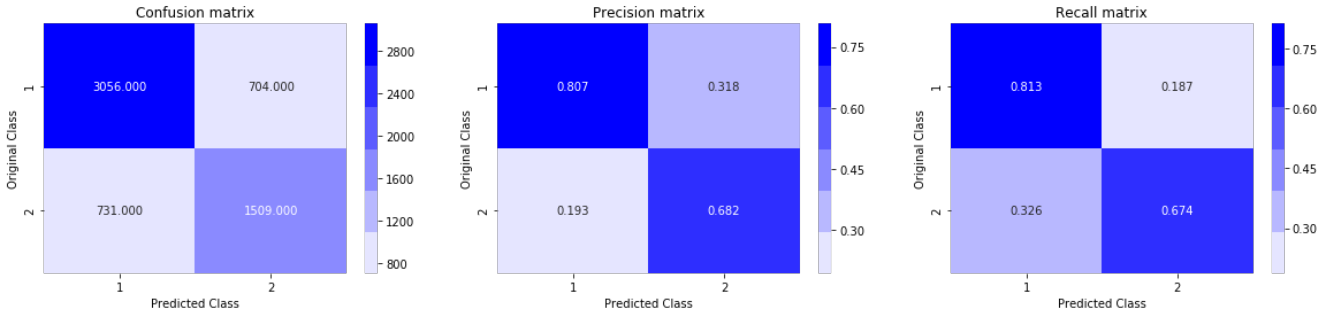
```
log_error_array=[]
for i in base_learn:
    clf = xgb.XGBClassifier(n_estimators = i,objective = 'binary:logistic',eval_metric = 'logloss',
eta = 0.02,max_depth = 4)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of base_learn = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=c
lf.classes_, eps=1e-15))

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

```
plt.ylabel("Error measure")
plt.show()


best_base_learn = np.argmin(log_error_array)
clf = xgb.XGBClassifier(n_estimators = base_learn[best_base_learn],objective =
'binary:logistic',eval_metric = 'logloss',eta = 0.02,max_depth = 4 )
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best base_learn = ', base_learn[best_base_learn], "The train log loss is:",lo
g_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best base_learn = ', base_learn[best_base_learn], "The test log loss is:",log
_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of base_learn =  30 The log loss is: 0.45242326880770384
For values of base_learn =  40 The log loss is: 0.4493511060325324
For values of base_learn =  50 The log loss is: 0.44765847857133295
For values of base_learn =  60 The log loss is: 0.44638462889253955
For values of base_learn =  70 The log loss is: 0.44589820716408457
For values of base_learn =  80 The log loss is: 0.4457846101029815
For values of base_learn =  90 The log loss is: 0.44583655838410036
For values of base_learn =  100 The log loss is: 0.4453380663908674
For values of base_learn =  110 The log loss is: 0.44590082661756264
```




```
For values of best base_learn =  100 The train log loss is: 0.4063534804815736
For values of best base_learn =  100 The test log loss is: 0.4453380663908674
Total number of data points : 6000
```



In [96]:

```
new = ['XGBoost','TFIDF','n-estimators = 100',0.406,0.445]
results.loc[2] = new
```

Performance Table:

In [97]:

```
results
```

Out[97]:

	Model	Featuraization	Hyperparameter	Train_loss	Test-loss
0	Logistic-Regession	TFIDF	alpha = 0.0001	0.559	0.559
1	Linear SVM	TFIDF	alpha = 10e-05	0.555	0.555
2	XGBoost	TFIDF	n-estimators = 100	0.406	0.445

Conclusion:

1.XGBoost with tfidf featuraization and hyper parameter tuning gives best results (log loss = 0.445) compared to logistic regression and Linear SVM

2.But time complexity is high compared to logistic regression and linear svm