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 (or slightly differently worded) of questions that have already been asked. This improves the customer experiance .
- This could be useful to merge the questions and instantly provide answers to questions that have already been answered.
- We are tasked with predicting whether a pair of questions are duplicates or not.

1.2 Sources/Useful Links

- Source : https://www.kaggle.com/c/quora-question-pairs (https://www.kaggle.com/c/quora-question-pairs) , we can get dataset and problem from this link.

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

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high. So we have to minimize it.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. Here we should get a probability that, given Que 1 is similar to Que 2 and probability lies between 0 and 1. If the probabaility is greater than choosen threshold probability, then merge the answers for those questions.
4. No strict latency concerns.
5. Interpretability is partially important to understand why questions are merged

Quora question pair productionization:-

1. Given a pair of questions we determine whether they are duplicates or not .

In productionizations,

1. There will be millions of questions and if we compare the question with these million questions, then we have to evaluate a million test points which are pairs of questions and it is very time taking.
2. So to solve this problem, given a new question, instead of comparing it with all the million questions, we can decrease the set of questions we should compare it with.
3. We can use simple fast searching scheme tool, we can break up millions of questions in database into list of most important words through similar word compitition through inverted indices.
4. Google does this process of inverted indixes . It searhces similar questions using algorithms called inverted indixes. Inverted indexes are most used in text search systems in the world and it is very very fast.
5. inverted indixes method is depends on key words. So we try to find the particular key word in all the questions in database, for a particular given question and for matched words we give more importance .
6. There are variations of inverted indices like distributed inverted indixes etc., depending on dataset size. So with process, instead of comparing the question with all the questions in database, now we have subset of questions which is smaller and we compare the given question with this subset.
7. This method gradually reduces the question comparision from million comparisions to just few comparisions, by just comparing keywords.
8. This is a optimization hack. There is no low latency requirments

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

1. Data will be in a file Train.csv
2. Train.csv contains 5 columns : qid1, qid2, question1, question2, is_duplicate
. is_dupliate is Yi here which belongs to {0,1}, and using qid1 ,qid2, question1, question2 we can construct Xi. We have to find whether the question is duplicate or not. 3. If is_duplicate = 0, that means the questions are not duplicates. If is_duplicate = 1, then questions are duplicates.
4. Size of Train.csv - 60MB
5. Number of rows in Train.csv = 404,290

Class 0 = questions are not duplicates

Class 1 = questions are duplicates

2.1.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

1. It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.
2. We have to featurize the data to determine whether the questions are duplicate or not. This is binary classification problem.

2.2.2 Performance Metric

Source: ***[https://www.kaggle.com/c/quora-question-pairs#evaluation (https://www.kaggle.com/c/quora-question-pairs#evaluation)](https://www.kaggle.com/c/quora-question-pairs#evaluation)***

Metric(s):

- log-loss : [https://www.kaggle.com/wiki/LogarithmicLoss (https://www.kaggle.com/wiki/LogarithmicLoss)](https://www.kaggle.com/wiki/LogarithmicLoss) . It is the metric for this problem. Here though it is a binary classification problem, we dont want output to be just 0 or 1.
- We want probability value which lies between 0 to 1 , to determine whether the question is duplicate or not. So when we take probability value, then log loss is one of the best metric.
- Log-loss is the primary KPI - Key Performance Indicator
- Binary Confusion Matrix . This is secondary performance metric, where we can compute precision , recall, TPR, TNR, FPR, FNR

2.3 Train and Test Construction

1. 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.
2. The model is built based on present day questions. If we give any new question for this model, then it determines whether this question is duplicate of already answered question.
3. Type of Questions asked change over time. And model also should change over time. So if we had a time stamp for each of question pairs then we can break the dataset based on time axis and sorted based on time and take the oldest 70% data as train data and newest 30% as test data. And this best way to split due to data change over time.
4. But here dont have time stamp and so we can't do temporal splitting.
5. So we can just randomly split train and test data.

3. Exploratory Data Analysis

In [4]:
```
# LOading the libraries
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 plotly.plotly as py
import os
import gc

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

# Loading libraries
# https://pypi.org/project/fuzzywuzzy/
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-pyt
hon3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image

from sklearn.preprocessing import MinMaxScaler
from sklearn.manifold import TSNE
import plotly
import plotly.plotly as py
```

3.1 Reading data and basic stats

In [5]:
```
# Loading the csv fine data to a dataframe
#df = pd.read_csv("train.csv")
df = pd.read_csv('train.csv')
print("Number of data points:",df.shape[0])
```

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

In [6]:
```
# printing top 5 rows in the file
# qid1 is question id of question 1.
# qid2 is question id of question 2
# is_duplicate is class label with 2 classes {0,1}

df.head()
```

Out[6]:

	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 [7]:
```
# Prints information of all columns
# question 2 has 2 null objects as it has 2 points less than total points.
# question 1 has 1 null value.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

Observation:-

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

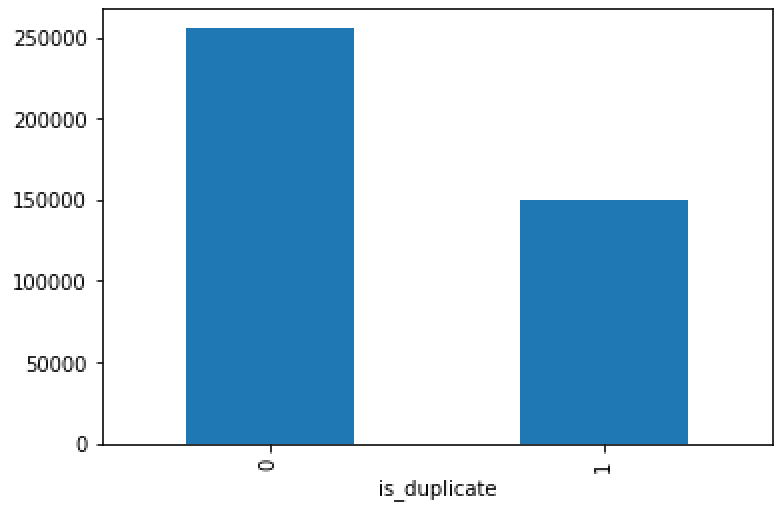
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other or not.

3.2.1 Distribution of data points among output classes

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

In [14]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba70ce3f28>
```



In [10]:
```
print('-> Total number of question pairs for training:\n   {}'.format(len(df
)))
```

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

In [11]:
```
# checking the percentage of duplicate and non duplicate questions
print('-> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(1
00 - round(df['is_duplicate'].mean()*100, 2)))
print('\n-> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(rou
nd(df['is_duplicate'].mean()*100, 2)))
```

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

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

3.2.2 Number of unique questions

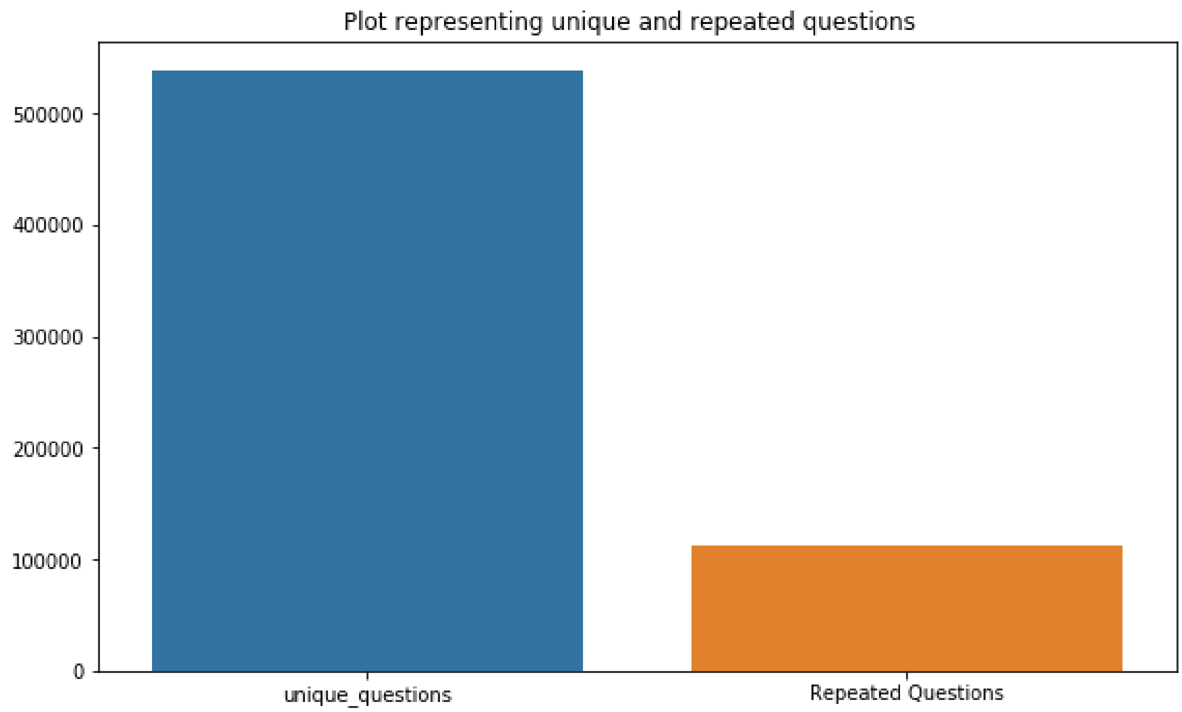
In [12]:
```
# Each of data points is a pair of questions (Q1,Q2) with class label 0 or 1
 (1 = similar/duplicate, 0 = not duplicate),
# Q1 can repeat with other combination of questions , Q2 also can repeat with
 combination of other questions.

qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
# unique questions
unique_qs = len(np.unique(qids))
# unique questions which appears more than once
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/once: {} ({}
%)\n'.format(qs_morethan_onetime, qs_morethan_onetime/unique_qs*100))

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time/once: 111780 (20.77
953945937505%)

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

Observations:-

1. From the above output we can say that there are 537933 unique questions
2. Number of unique questions that appear more than one time are 111780 that is 20% of total questions. 80% of remaining questions occur only once. That means majority of questions occur only once.
3. There is one question that appears 157 times , which is the largest number of times any question occurs .

In [13]:
```
# Plot which shows how many unique questions are there and how many number of
 repeated questions.
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()
```



Observations:-

1. We can observe 80% are unique questions and 20% are repeated questions

3.2.3 Checking for Duplicates

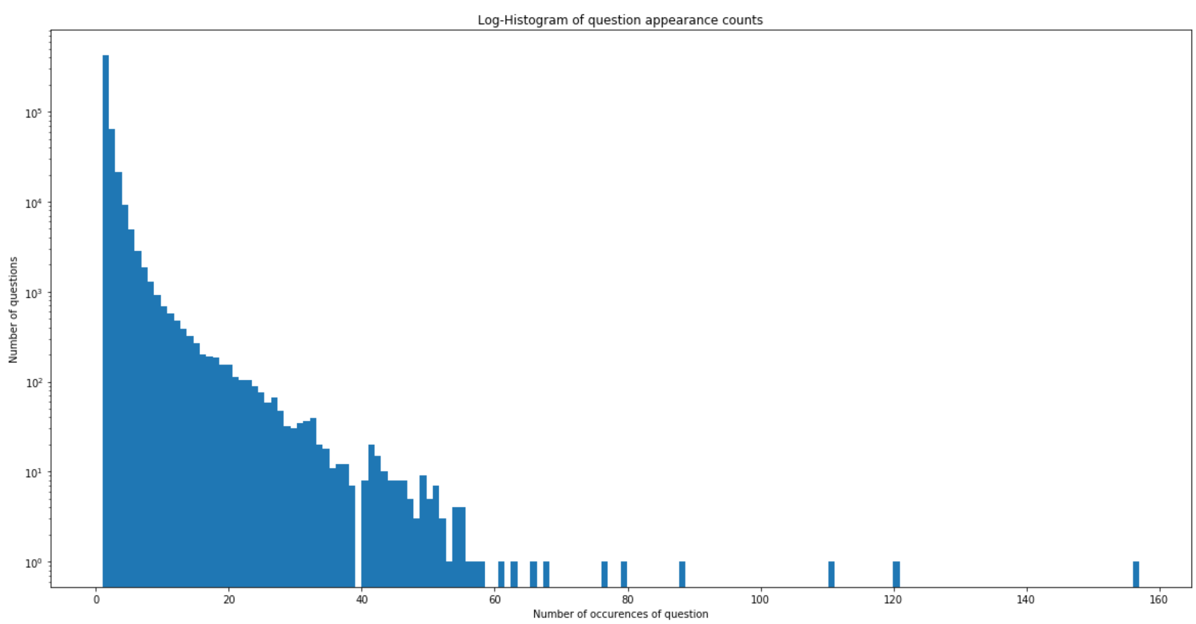
In [15]:
```
#checking whether there are any repeated pair of questions./ duplicate rows
# Same questions can be paired with different questions then it is not repeate
d pair

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

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

```
Number of duplicate questions pairs 0
```

3.2.4 Number of occurrences of each question

In [16]:
```
# This is like histogram  plot
# Y-axis is logarithmic axis where 10^0 = 1, 10^1= 10, 10^2=100, 10^3=1000, 10
^4=10000, 10^5=100000

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

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

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

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

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

plt.ylabel('Number of questions')

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

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



Observations:-

1. Number of questions which occur only once are maximum, 80% of questions don't repeat.
2. There is one que that occurs 157 times which is highest.
3. Most of the questions occur between 0 to 40 occurances.

3.2.5 Checking for NULL values

In [17]:
```
#Checking whether there are any rows with null values. There are 2 questions w
hich have null/NAN
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

```
            id    qid1    qid2                         question1  \
105780  105780  174363  174364    How can I develop android app?
201841  201841  303951  174364  How can I create an Android app?
363362  363362  493340  493341                               NaN

                                                question2  is_duplicate
105780                                                NaN             0
201841                                                NaN             0
363362  My Chinese name is Haichao Yu. What English na...             0
```

Observation:-

1. There are two rows with null values in question2
2. There is one row with null values in question 1

In [18]:
```
# For cleaning up of null values or NAN values, we do Filling / Replacing the
 null values with ' ' empty string.
# so that there are no null values

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)

1. This is high-level basic feature extraction

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's . frequency means number of times a question occurs
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1 . This is string length of questions
- **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) . If there are more commoon words then questions can be duplicate.
- **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 [19]:
```
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[19]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65	
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	

3.3.1 Analysis of some of the extracted features

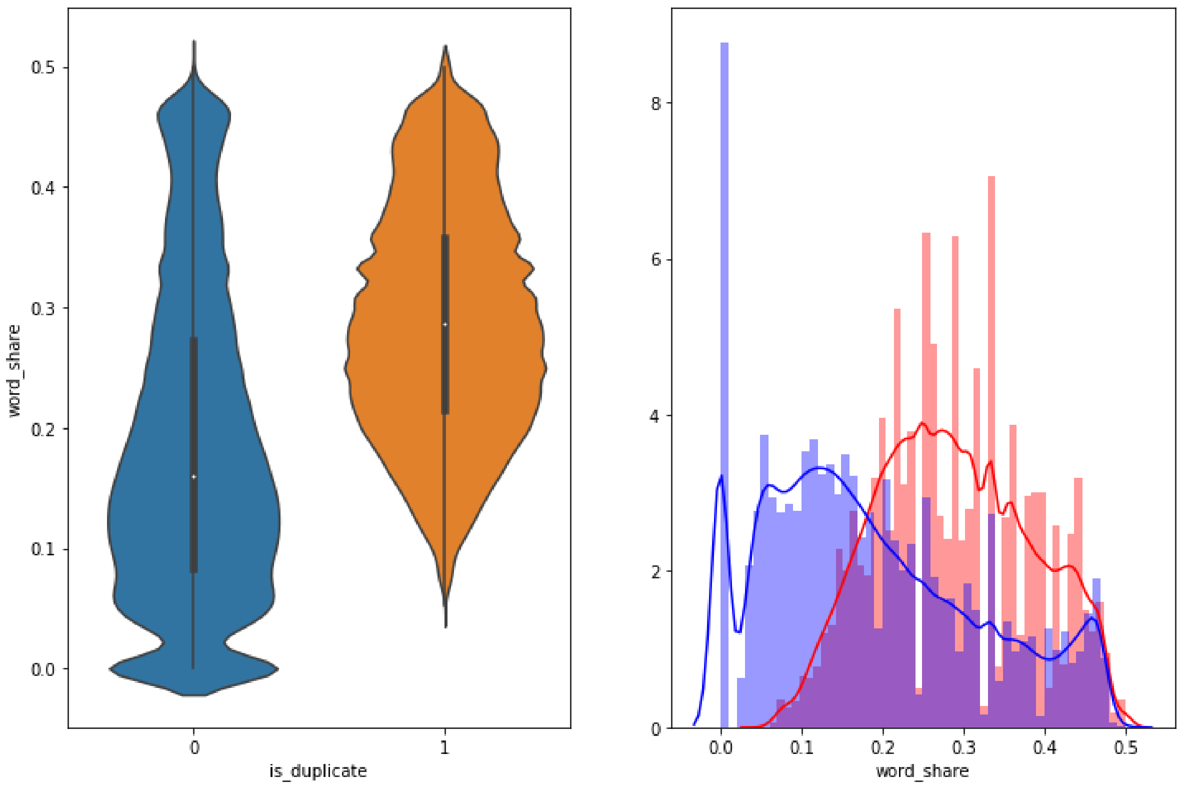
- Here are some questions have only one single words.

In [20]:
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words'
]))

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

print ("Number of Questions with minimum length [question1] :", df[df['q1_n_wo
rds']== 1].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_wo
rds']== 1].shape[0])
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

In [21]:
```
# 1 = duplicates
# 0 = not duplicates
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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



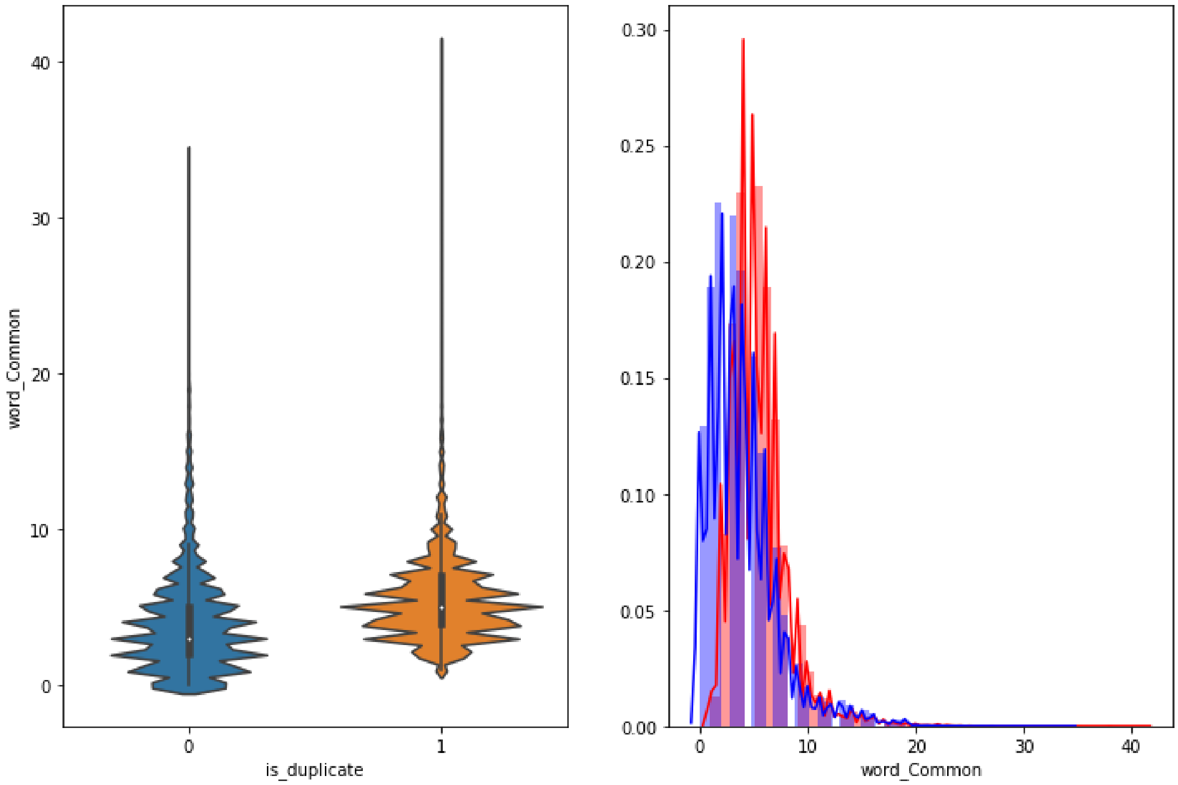
Observations :-

1. This plot is the distribution of 2 classes {0,1} and there 2 types of plots - PDF's and Violin plot . blue color = word share of class 0 , red color = word share of class 1.
2. As the word_share increases there is higher chance that the questions are duplicates. The less the overlap between the 2 distributions , the better is the feature.
3. There is overlap between the points in PDF plot. Box plots in violin plot are not perfectly overlapping , so it has some values to differentiate 2 classes.

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

3.3.1.2 Feature: word_Common

In [22]:
```
plt.figure(figsize=(12, 8))

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

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



Observations:-

Blue = class 0 , red = class 1

1. The PDF plot is harder to interpret and violin plots are better.
2. If there is more overlap between the boxplots in violin plot, the worse the feature is. There is more overlap between the 2 box plots in violin plot.
3. The distributions of the word_Common feature in similar and non-similar questions are highly overlapping .

EDA: Advanced Feature Extraction

In [23]:
```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-ca
nt-decode-byte-0x9c
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-
1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the pre
vious notebook")
```

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

Out[24]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	

3.4 Preprocessing of Text

Preprocessing:

1. Removing html tags . To remove html tags we can use Regular expressions
2. Removing Punctuations
3. Performing stemming
4. Removing Stopwords
5. Expanding contractions etc. Eg:- In place of '%' symbol we place with 'percent' , "he's" is replaced by 'he is' and so on.

Porter stemmer is popular stemming algorithms. It produces morphological variants of a root/base word.

A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

In [28]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [29]:
```
# To get the results in 4 decemal points
# Beautiful soup is used to Remove all tags from a string
# reference -- https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
import warnings
warnings.filterwarnings("ignore")
import nltk
nltk.download('stopwords')
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    # m = million , k = thousand   , 000000 is replaced with 'm' and 000 is replaced with 'k'
    x = x.replace(",000,000", "m").replace(",000", "k").replace("′", "'").replace("’", "'")\
                           .replace("won't", "will not").replace("cannot", "can not").replace("can't", "can not")\
                           .replace("n't", " not").replace("what's", "what is").replace("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").replace("'s", " own")\
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar ")\
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)   # regular expressions
    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
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/pavana_paradesi5/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

- **Token**: You get a token by splitting sentence by a space
- **Stop_Word** : stop words as per NLTK. Eg:- of, if,. etc
- **Word** : A token that is not a stop_word

1. CWC- Common Word Count , CSC - Common Stopword Count

Features:

There are total of 15 features

word is a token but not a stop word. So we take common words in both questions and take common word count and divide it by minimum of length of q1 words and w2 words

- **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))

Here we take stop word count

- **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))

We take token count. we get token by breaking string with spaces

- **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 equal

If the last word in both the questions is same then it returns 1 otherwise 0. It is a boolean feature

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

First word equal

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

Absolute length difference

It is defined as absolute value of the difference between the length of q1 tokens and q2 tokens

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

Mean length

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

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

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

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

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

Longest substring ratio:-

1. Longest common substring/tokens is taken from both questions / min of number of tokens in both questions.

- **longest_substr_ratio** : Ratio of length longest common substring in both sentences/strings/questions to min lenghth of token count of Q1 and Q2.

  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

FuzzyWuzzy

1. The above defined 4 methods/strategies , fuzz_ratio, fuzz_partial_ratio, token_sort_ratio , token_set_ratio are used to measure similarity between 2 sentences / questions. These strategies come up with similarity function
2. Similarity lies between 0 to 100 . 100 = sentences are very similar . 0 = sentences are dissimilar / not similar.
3. Edit Distance can do multiple operations like adding a alphabet , delete an alphabet or shift an alphabet.
4. Edit distance between the 2 sentences is small if we can convert one sentence to other sentence by making very few edits.
5. If the number of Edit operations between 2 sentences are very few, then the similarity between the 2 sentences will be high.

Fuzz_ratio :-

Eg:- fuzz_ratio ("NEW YORK METS", "NEW YORK MEATS")- 96

1. In this eg if we give 2 sentences to fuzz_ratio then it gives the similarity value. Value close to 100 means the sentences are similar almost
2. Fuzz_ratio can sometimes cannot give the correct similarities . It caanot solve some problems and can give different results.
3. So to solve problems in fuz__ratio, we define new metric called fuzz.partial_ratio.

Fuzz.partial_ratio

1. It looks for a perfect partial string / sub string that matches perfectly in given sentences . If there is matching ,Then it gives fuzz.partial_ratio value which is high. So strings should partially match in this case.
2. This also have issues called Out of Order , where it looks for order, if the order is mismatched then it gives less value.
3. So to overcome this problem there is another feature called token_sort_ratio .

fuzz.token_sort_ratio :-

1. It takes the sentence , breaks into individual words and sort them in alphabetical order . Then give these changed sentences to fuzz.token_sort_ratio , then it gives high similarity value .
2. THis also have some problems, so there comes token_set_ratio

fuzz.token_set_ratio

1. This takes sorted intersection of words in sentences.
2. So the process of this is, First we take sentences and so token sort and get new tokens T1,T2, then we compute token set t0,t1,t2

t0 = sorted_intersection
t1 = sorted_intersection + sorted rest of words in String1 (remaining words)
t2 = sorted_intersection + sorted rest of words in String2

Next we can compute fuzz_ratio of (t0,t1) , (t0,t2) , (t1,t2). fuzz_ratio of all possibilities is taken, and maximum value of these is taken and it is token_set_ratio value.

All the above features are important to compute similarity between 2 sentences.

In [30]:
```
def get_token_features(q1, q2):
    token_features = [0.0]*10

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

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

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

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

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

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


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words))
+ SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words))
+ SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops))
+ SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops))
+ SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_token
s)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_token
s)) + SAFE_DIV)

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

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

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

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

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
```

```
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

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

    print("token features...")

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["q
uestion1"], x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sort
ing the tokens alphabetically, and
    # then joining them back into a string We then compare the transformed str
ings with a simple ratio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x[
"question1"], x["question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"
], x["question2"]), axis=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["que
stion1"], x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(
x["question1"], x["question2"]), axis=1)
    return df
```

In [32]:
```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

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

Out[32]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988

2 rows × 21 columns

3.5.1 Analysis of extracted features

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
- Word clouds are plotted to find specific words that we find more often in duplicate questions and less often in non-duplicate questions.
- Some words occur more often in class 1 as compared to class 0 and vice versa

In [33]:
```
# https://stackoverflow.com/questions/49684095/python-unicodeencodeerror-charm
ap-codec-cant-encode-characters-in-position

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

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

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

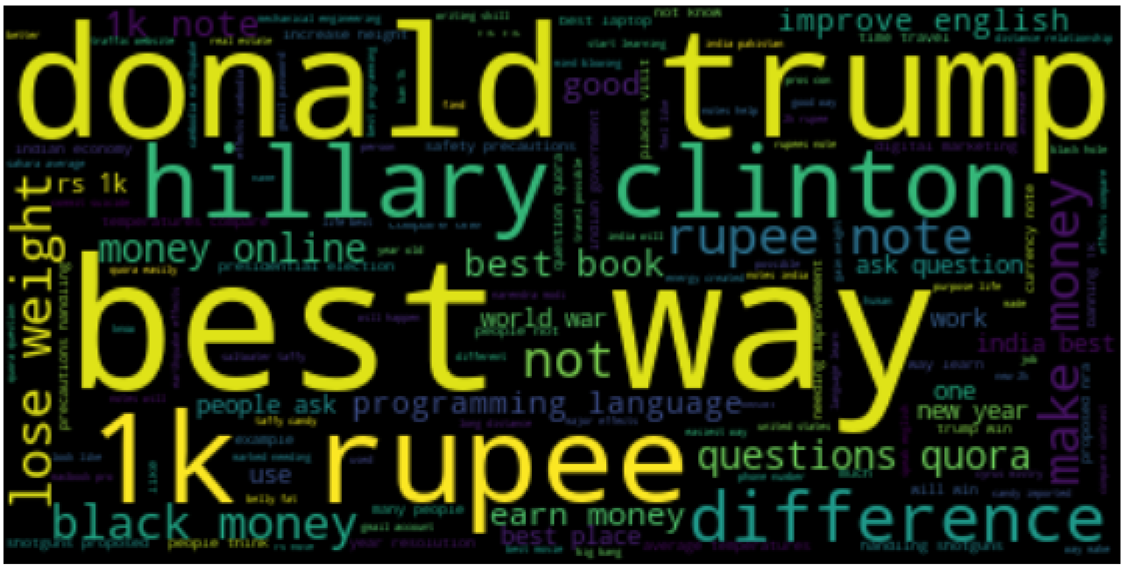
In [34]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

# reference - https://stackoverflow.com/questions/27092833/unicodeencodeerror-
charmap-codec-cant-encode-characters
# the encoding is changed to UTF-8 when using the file, so characters in UTF-8
are able to be converted to text,
# instead of returning an error when it encounters a UTF-8 character that is n
ot supported by the current encoding.
# ,encoding='utf-8'
textp_w = open(path.join(d, 'train_p.txt'),encoding='utf-8').read()
textn_w = open(path.join(d, 'train_n.txt'),encoding='utf-8').read()
stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

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

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193067
```

Word Clouds generated from duplicate pair question's text

In [35]:
```
# 1 = duplicate pairs

wc = WordCloud(background_color="black", max_words=len(textp_w), stopwords=sto
pwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.figure(figsize=(10,10))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

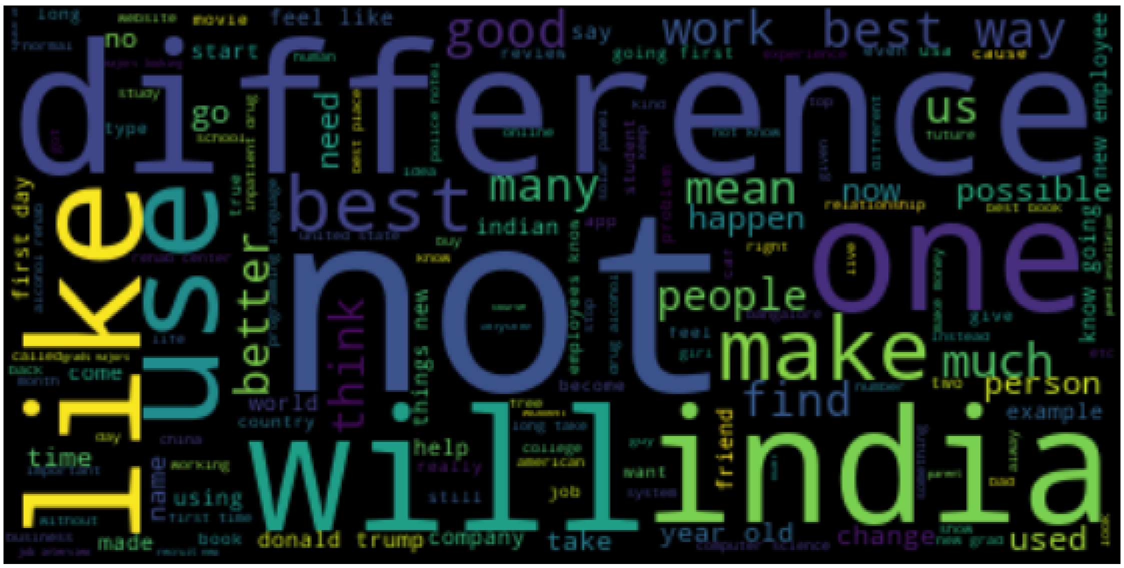
```
Word Cloud for Duplicate Question pairs
```



Word Clouds generated from non duplicate pair question's text

In [36]:
```python
# 0 = non-duplicate pairs
wc = WordCloud(background_color="black", max_words=len(textn_w),stopwords=stop
words)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.figure(figsize=(10,10))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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



Observation:-

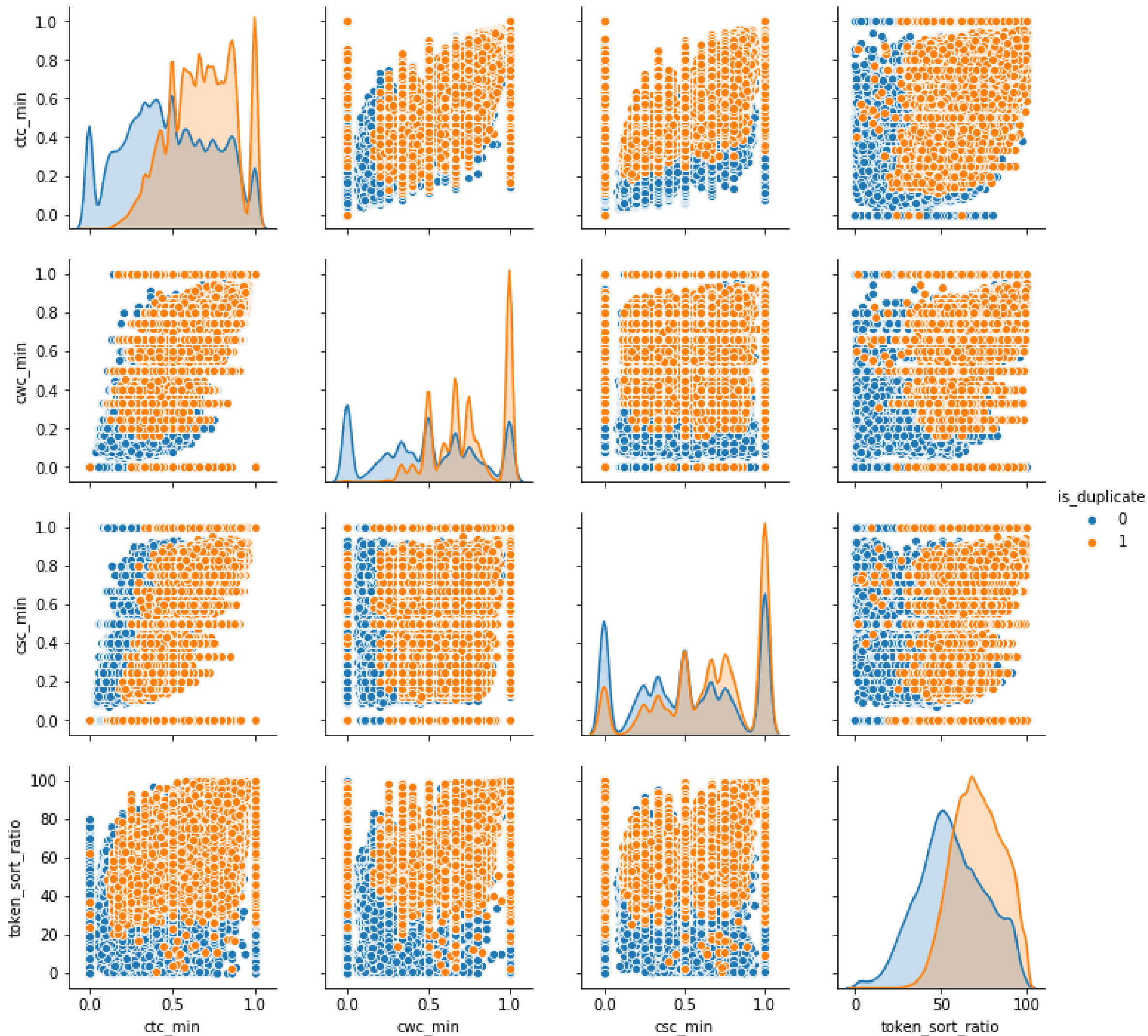
1. Some words occur more often in class 1 as compared to class 0 and vice versa.
2. We can use TFIDF to count words

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

0 = not duplicates

1 = duplicates

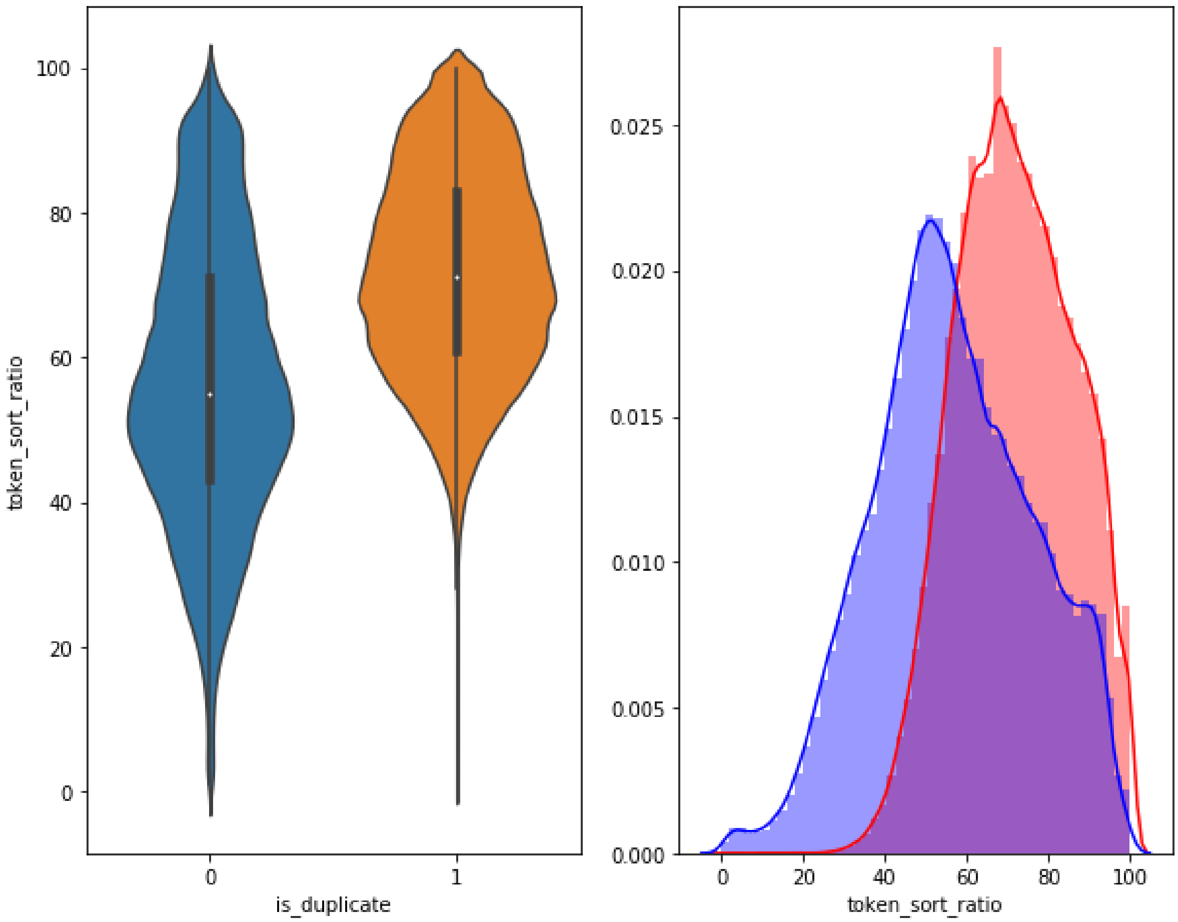
In [38]:
```
# These are pair plots  , orange color = class 1 , blue color = class 0
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```



Observation:-

1. In ctc_min , there are more of class = 1 occuring . csc_min also partially seperated between the classes 1 and 0.
2. token_sort_ratio and csc_min also seperates classes well.
3. All the features are useful as univariate features (single features) and bivariate features(pairs)

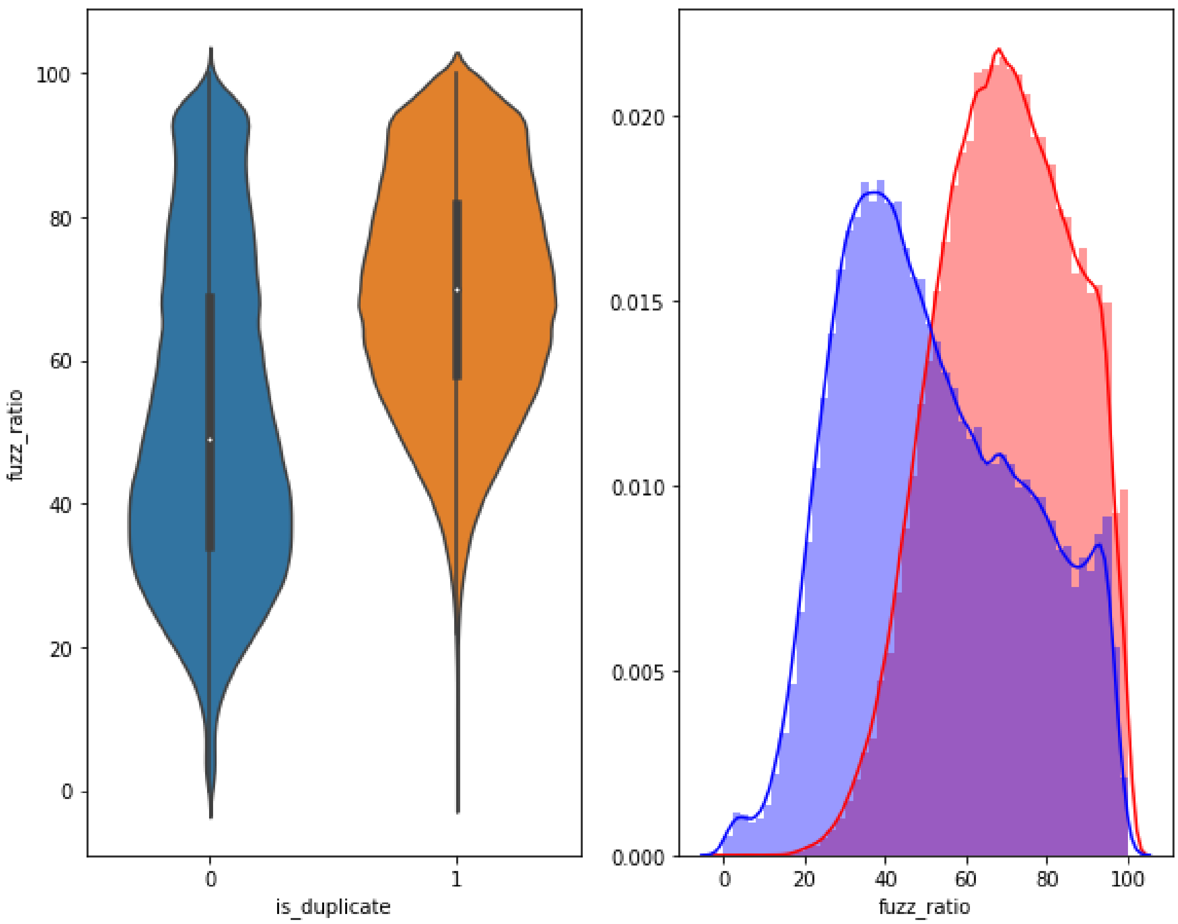
In [39]:
```
# Distribution of the feature token_sort_ratio . blue color = class 0 , orange color = class 1
plt.figure(figsize=(10, 8))

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()
```



Observation :-

1. In the above PDF ,there is overlap between points of both classes 0 and 1, but class 1 points have larger value of token_sort_ratio than class 0 points
2. In the violin plot, the box plots are not fully overlapping each other , so token_sort_ratio is a important feature for classification.

In [40]:
```
# Distribution of the feature - fuzz_ratio
plt.figure(figsize=(10, 8))

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

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



Observation:-

1. Here also there is overlap between the classes but more points of class 1 have larger value of fuzz_ratio than points belonging to class 0
2. Violin plots are not fully overlapping . so feature fuzz_Ration is useful to determine the class label

Till here we have done univariate (trying to see in individual features are useful) and bi-variate analysis (pair plots to see if pairs of features are useful).

We can visualize data in all 15 features using techniques called T-SNE , to understand if in the high dim space of 15 features, if class 1 and class 0 points are seperated.

3.5.2 Visualization

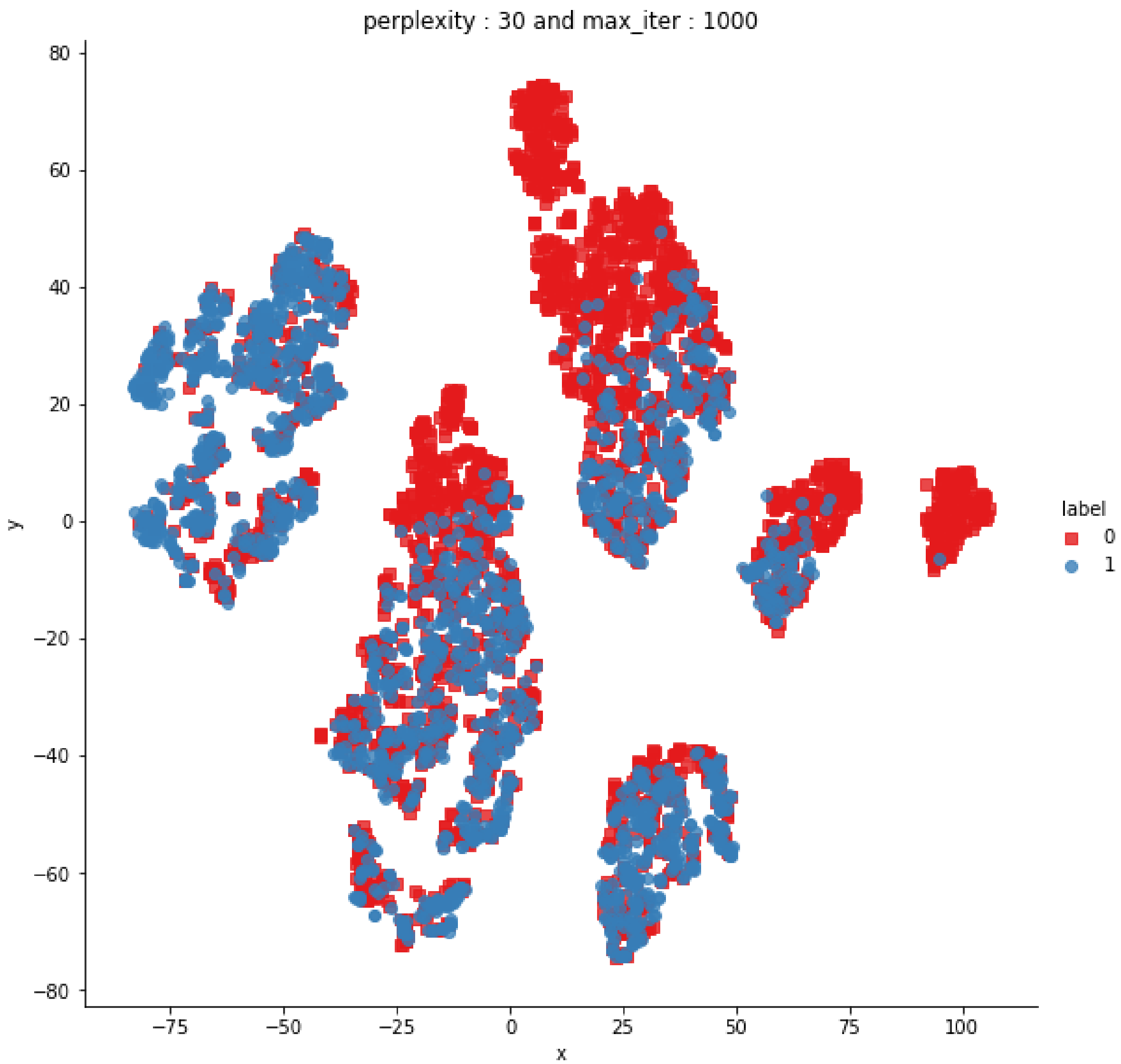
In [41]:
```
# Using TSNE for Dimentionality reduction of 15 Features(Generated after clean
ing the data) to 3 dimention
# Our data is in 15 dimensions, so using TSNE we can project this into 2 dim d
ataset and can visualize this 2D dataset using a plot


dfp_subsampled = df[0:5000]  # taken subset of points as TSNE takes lot of tim
e to run
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_mi
n', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_word_eq' , 'abs
_len_diff' , 'mean_len' , 'token_set_ratio' , 'token_sort_ratio' ,  'fuzz_rati
o' , 'fuzz_partial_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

In [42]:
```
# we take only 2 dim with random initialization, run for 1000 iterations
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 0.019s...
[t-SNE] Computed neighbors for 5000 samples in 0.362s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.346s
[t-SNE] Iteration 50: error = 81.2911148, gradient norm = 0.0457501 (50 itera
tions in 2.699s)
[t-SNE] Iteration 100: error = 70.6044159, gradient norm = 0.0086692 (50 iter
ations in 1.858s)
[t-SNE] Iteration 150: error = 68.9124908, gradient norm = 0.0056016 (50 iter
ations in 1.765s)
[t-SNE] Iteration 200: error = 68.1010742, gradient norm = 0.0047585 (50 iter
ations in 1.843s)
[t-SNE] Iteration 250: error = 67.5907974, gradient norm = 0.0033576 (50 iter
ations in 1.932s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.590797
[t-SNE] Iteration 300: error = 1.7929677, gradient norm = 0.0011899 (50 itera
tions in 1.944s)
[t-SNE] Iteration 350: error = 1.3937442, gradient norm = 0.0004817 (50 itera
tions in 1.916s)
[t-SNE] Iteration 400: error = 1.2280033, gradient norm = 0.0002773 (50 itera
tions in 1.919s)
[t-SNE] Iteration 450: error = 1.1383208, gradient norm = 0.0001865 (50 itera
tions in 1.941s)
[t-SNE] Iteration 500: error = 1.0834006, gradient norm = 0.0001423 (50 itera
tions in 1.941s)
[t-SNE] Iteration 550: error = 1.0474092, gradient norm = 0.0001144 (50 itera
tions in 1.931s)
[t-SNE] Iteration 600: error = 1.0231259, gradient norm = 0.0000995 (50 itera
tions in 1.961s)
[t-SNE] Iteration 650: error = 1.0066353, gradient norm = 0.0000895 (50 itera
tions in 1.983s)
[t-SNE] Iteration 700: error = 0.9954656, gradient norm = 0.0000805 (50 itera
tions in 1.995s)
[t-SNE] Iteration 750: error = 0.9871529, gradient norm = 0.0000719 (50 itera
tions in 2.012s)
[t-SNE] Iteration 800: error = 0.9801921, gradient norm = 0.0000657 (50 itera
tions in 2.007s)
[t-SNE] Iteration 850: error = 0.9743395, gradient norm = 0.0000631 (50 itera
tions in 2.006s)
[t-SNE] Iteration 900: error = 0.9693972, gradient norm = 0.0000606 (50 itera
tions in 2.025s)
[t-SNE] Iteration 950: error = 0.9654404, gradient norm = 0.0000594 (50 itera
tions in 2.017s)
[t-SNE] Iteration 1000: error = 0.9622302, gradient norm = 0.0000565 (50 iter
ations in 2.020s)
[t-SNE] KL divergence after 1000 iterations: 0.962230
```

TSNE Plot

In [43]:
```
# 1 = duplicates , 0 = not duplicates

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()
```



Observation:-

1. In this plot, we are visualizing 15 dim data as 2 dimensional data , and class 0 and class 1 points are clearly seperable in some cases.

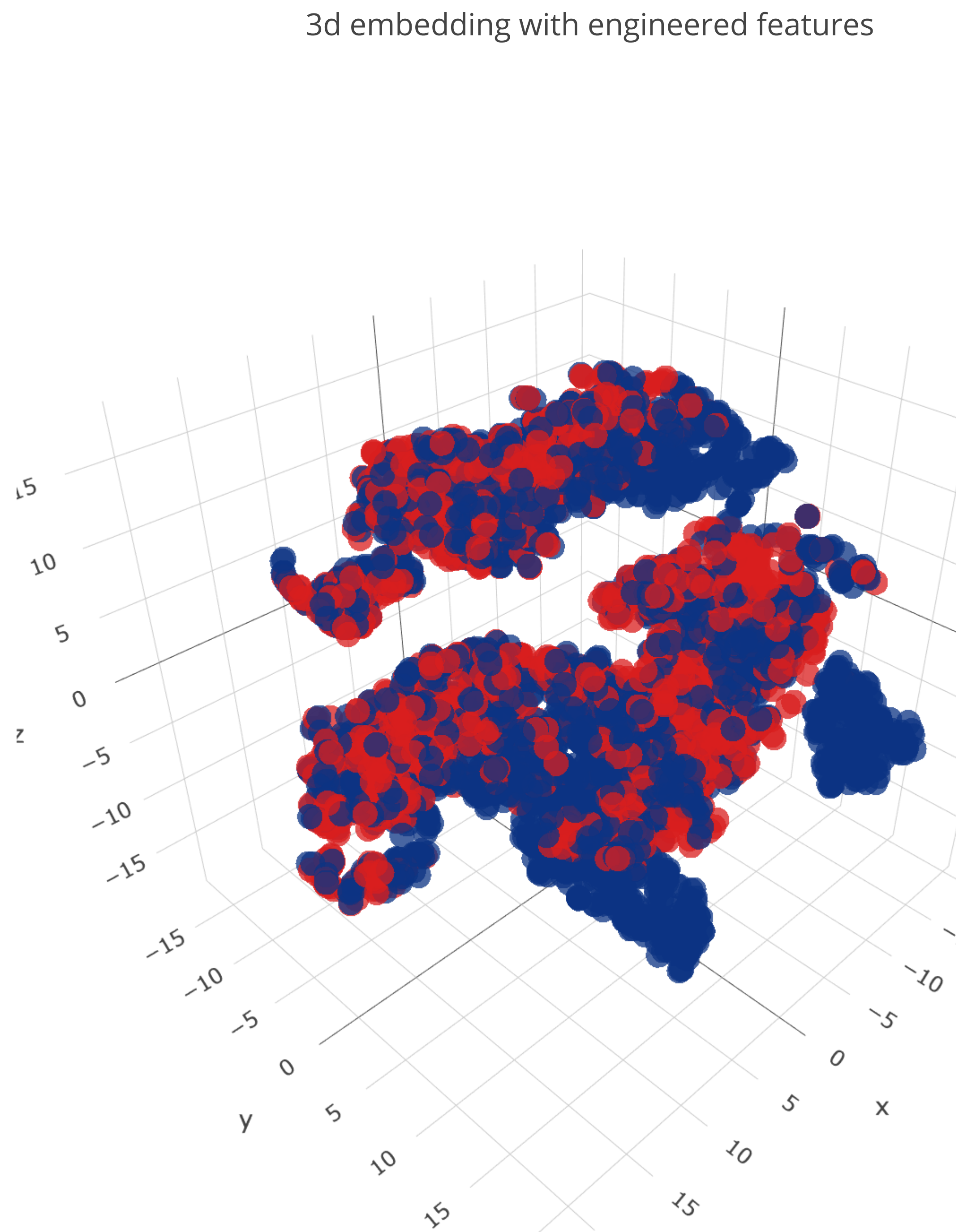
In [44]:
```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.009s...
[t-SNE] Computed neighbors for 5000 samples in 0.364s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.318s
[t-SNE] Iteration 50: error = 80.5316772, gradient norm = 0.0296611 (50 itera
tions in 13.809s)
[t-SNE] Iteration 100: error = 69.3834763, gradient norm = 0.0033837 (50 iter
ations in 8.621s)
[t-SNE] Iteration 150: error = 67.9741974, gradient norm = 0.0017825 (50 iter
ations in 8.003s)
[t-SNE] Iteration 200: error = 67.4170685, gradient norm = 0.0011107 (50 iter
ations in 8.013s)
[t-SNE] Iteration 250: error = 67.1046600, gradient norm = 0.0010527 (50 iter
ations in 8.020s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.104660
[t-SNE] Iteration 300: error = 1.5259259, gradient norm = 0.0007163 (50 itera
tions in 10.597s)
[t-SNE] Iteration 350: error = 1.1802596, gradient norm = 0.0001998 (50 itera
tions in 13.414s)
[t-SNE] Iteration 400: error = 1.0343297, gradient norm = 0.0000972 (50 itera
tions in 12.743s)
[t-SNE] Iteration 450: error = 0.9609142, gradient norm = 0.0000872 (50 itera
tions in 12.369s)
[t-SNE] Iteration 500: error = 0.9233505, gradient norm = 0.0000706 (50 itera
tions in 12.397s)
[t-SNE] Iteration 550: error = 0.9047915, gradient norm = 0.0000438 (50 itera
tions in 12.342s)
[t-SNE] Iteration 600: error = 0.8910298, gradient norm = 0.0000373 (50 itera
tions in 12.348s)
[t-SNE] Iteration 650: error = 0.8802577, gradient norm = 0.0000349 (50 itera
tions in 12.343s)
[t-SNE] Iteration 700: error = 0.8713953, gradient norm = 0.0000309 (50 itera
tions in 12.320s)
[t-SNE] Iteration 750: error = 0.8650095, gradient norm = 0.0000374 (50 itera
tions in 12.332s)
[t-SNE] Iteration 800: error = 0.8615507, gradient norm = 0.0000353 (50 itera
tions in 12.277s)
[t-SNE] Iteration 850: error = 0.8587439, gradient norm = 0.0000269 (50 itera
tions in 12.083s)
[t-SNE] Iteration 900: error = 0.8559932, gradient norm = 0.0000416 (50 itera
tions in 12.121s)
[t-SNE] Iteration 950: error = 0.8541381, gradient norm = 0.0000238 (50 itera
tions in 12.379s)
[t-SNE] Iteration 1000: error = 0.8511153, gradient norm = 0.0000228 (50 iter
ations in 12.331s)
[t-SNE] KL divergence after 1000 iterations: 0.851115
```

In [45]:
```
# https://github.com/plotly/plotly.py/issues/860
# https://plot.ly/~pavanap/0/_3d-embedding-with-engineered-features/#/

import plotly
import plotly.plotly as py
plotly.tools.set_credentials_file(username='pavanap', api_key='yKX7Kw2bjiLDUiF
Z3xXA')

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

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

```
High five! You successfully sent some data to your account on plotly. View yo
ur plot in your browser at https://plot.ly/~pavanap/0 or inside your plot.ly
account where it is named '3DBubble'
```

Out[45]:



In [ ]: