Quora Question Pairs

1. Business Problem

1.1 Description

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

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

Credits: Kaggle

Problem Statement

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

1.2 Sources/Useful Links

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

Useful Links

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

```
! pip install distance
```

```
Requirement already satisfied: distance in /usr/local/lib/python3.6/dist-packages (0
```

```
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 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

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

```
Choose Files train.csv
• train.csv(application/vnd.ms-excel) - 63399110 bytes, last modified: 6/11/2018 - 100% done
Saving train.csv to train (1).csv
User uploaded file "train.csv" with length 63399110 bytes
```

```
import pandas as pd
```

```
import io
df = pd.read_csv(io.StringIO(uploaded['train.csv'].decode('utf-8')))
```

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

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

```
df.head()
```

	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	Find the remainder when	0

```
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
```

```
df.shape
```

```
(404290, 6)
```

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

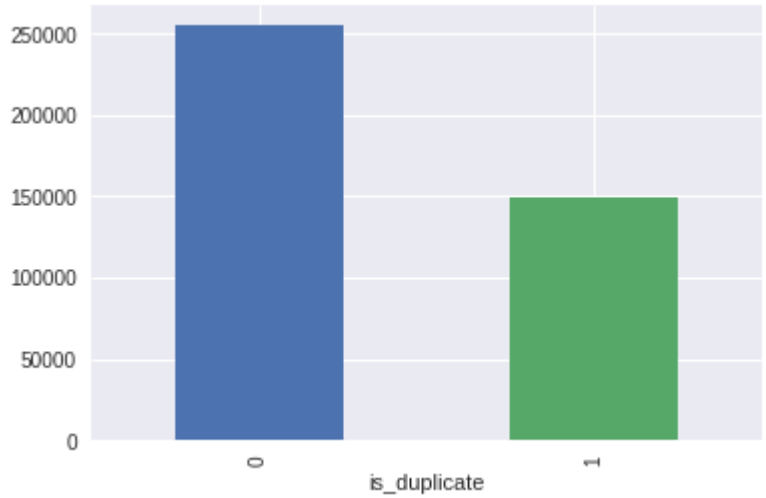
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa09c026630>
```




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

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

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

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

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

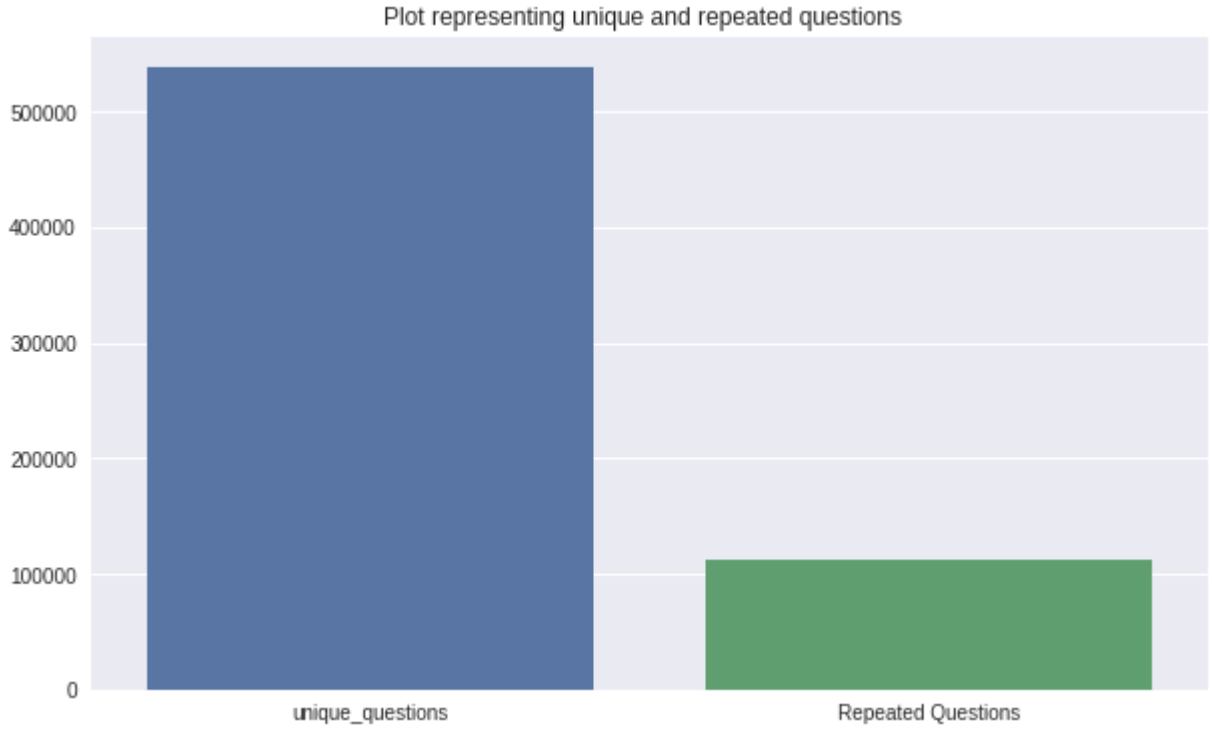
3.2.2 Number of unique questions

```
import numpy as np
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_mo

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.77953945937505

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:1428: FutureWarning: r
  stat_data = remove_na(group_data)
```



3.2.3 Checking for Duplicates

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

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

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

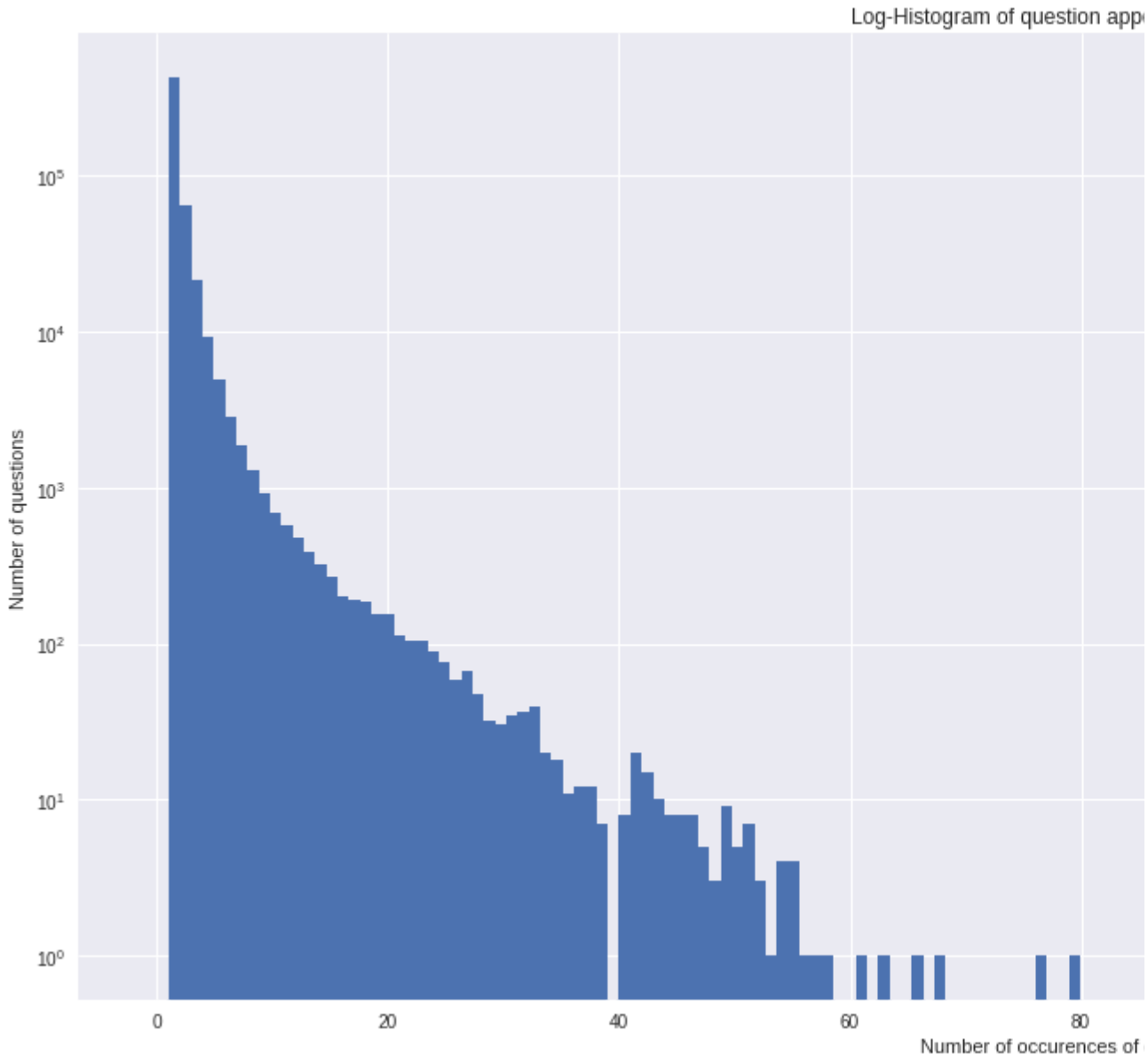
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

```
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_
```

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



3.2.5 Checking for NULL values

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

```
                id    qid1    qid2                         question1  \
105780  105780  174363  174364    How can I develop android app?
```

- There are two rows with null values in question2

```
# 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

```
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()
```

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

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

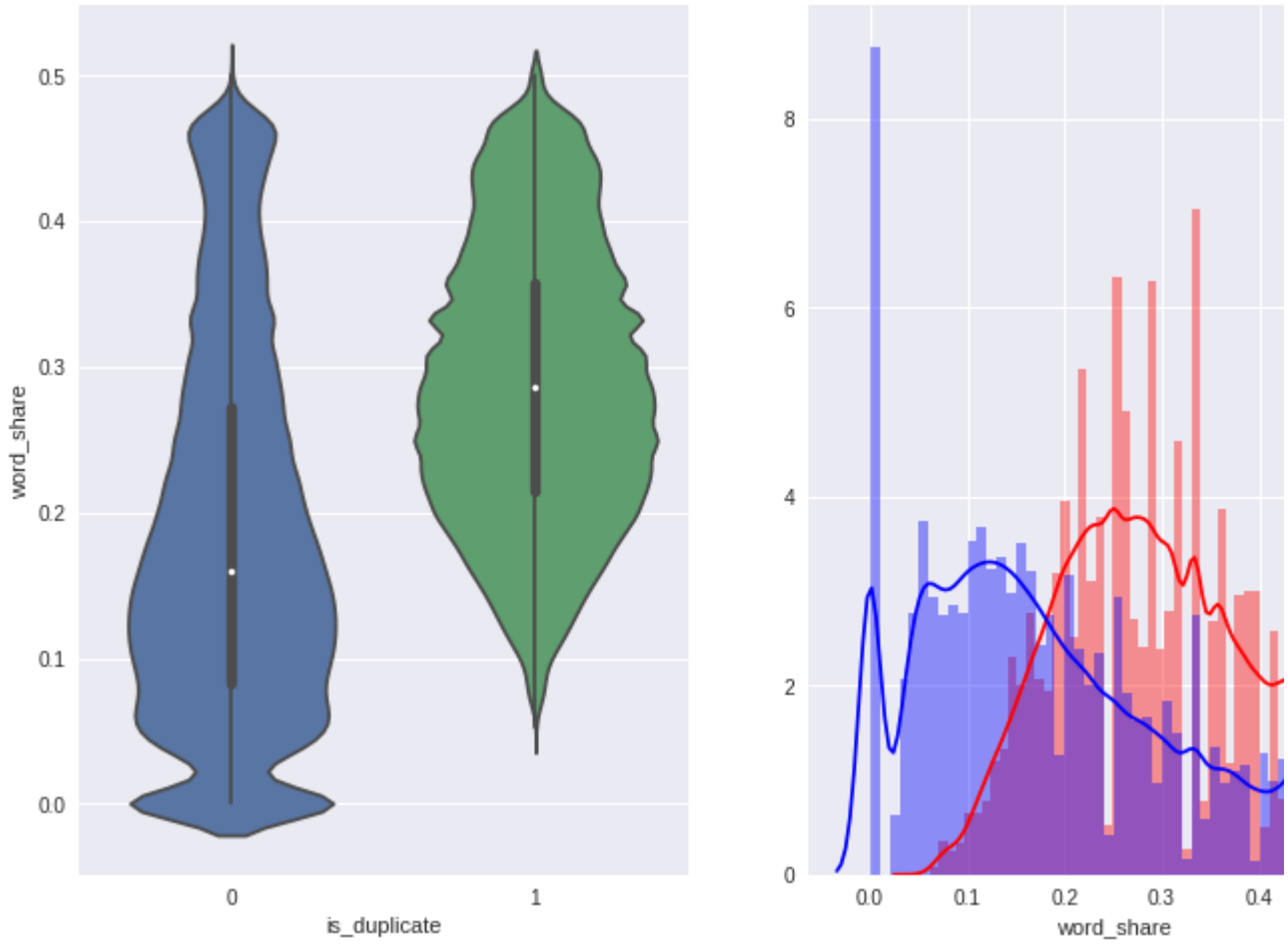
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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].sh
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']== 1].sh
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
```

3.3.1.1 Feature: word_share

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: re
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: re
  violin_data = remove_na(group_data)
```

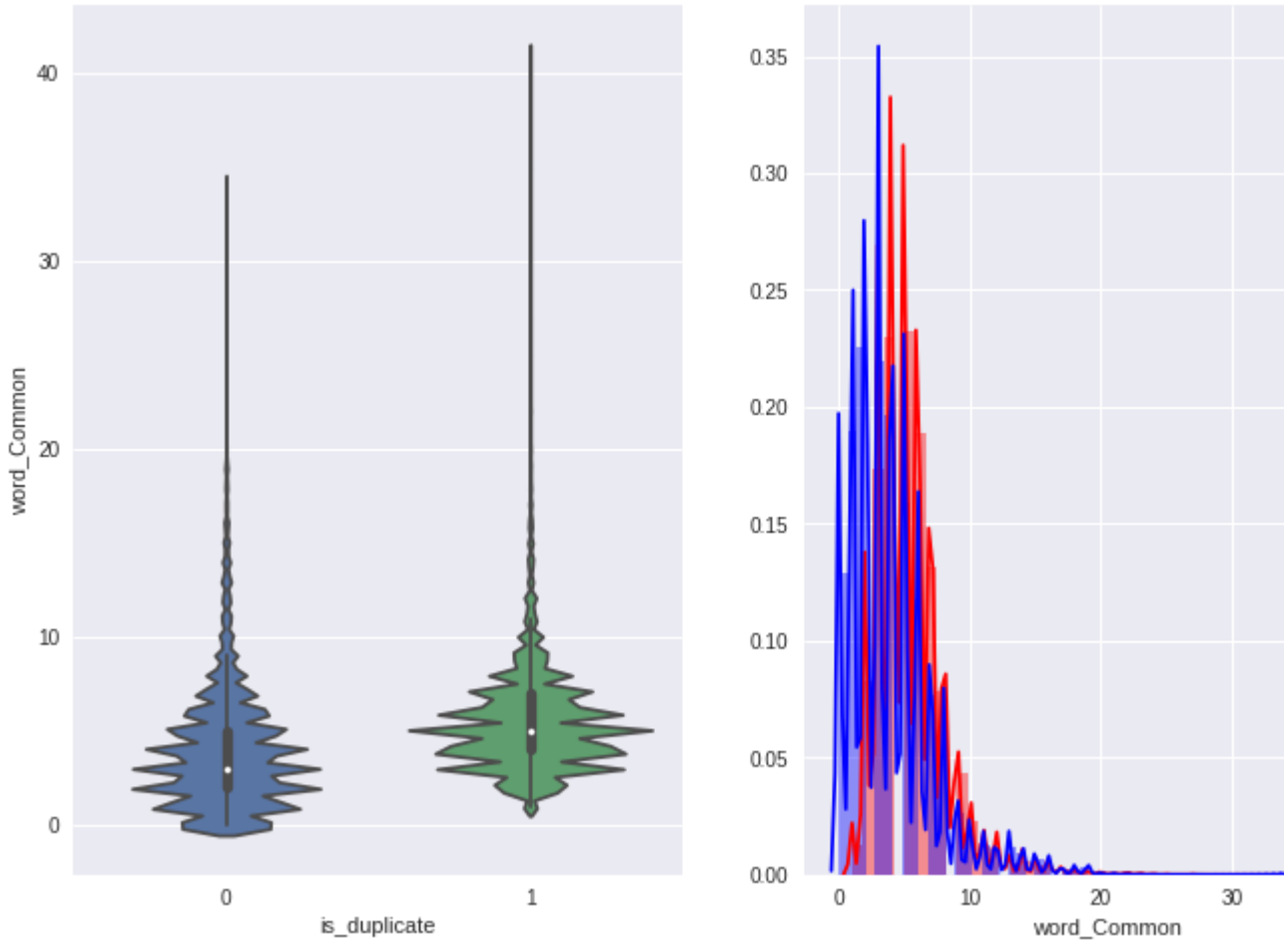


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: re
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: re
  violin_data = remove_na(group_data)
```




```
! pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/3b/36/be990a35c7e8ed9dc176c43b
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.16.0
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
```

```
[nltk_data] Downloading package stopwords to /content/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
/usr/local/lib/python3.6/dist-packages/fuzzywuzzy/fuzz.py:35: UserWarning: Using slo
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein
```

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

```
# 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").repla
                           .replace("n't", " not").replace("what's", "what is").replace("it
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re", "
                           .replace("he's", "he is").replace("she's", "she is").replace("'s
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$",
                           .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



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_DI
        token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DI

        # 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"]),

        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-
        # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-function-to-compa
        # https://github.com/seatgeek/fuzzywuzzy
        print("fuzzy features..")

        df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"], x
        # The token sort approach involves tokenizing the string in question, sorting the token
        # then joining them back into a string We then compare the transformed strings with a s
        df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"],
        df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["questio
        df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"], x["
```

```
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["question1"
    return df
```

```
df.shape
```

```
(404290, 17)
```

```
df = extract_features(df)
df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

```
token features...
fuzzy features..
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len
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
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

2 rows × 32 columns

```
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')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s')
```

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

```
# reading the text files and removing the Stop Words:
from os import path
from PIL import Image
d = path.dirname('.')

textp_w = open(path.join(d, 'train_p.txt')).read()
textn_w = open(path.join(d, 'train_n.txt')).read()
stopwords = set(STOP_WORDS)
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
```

```
! pip install wordcloud
```

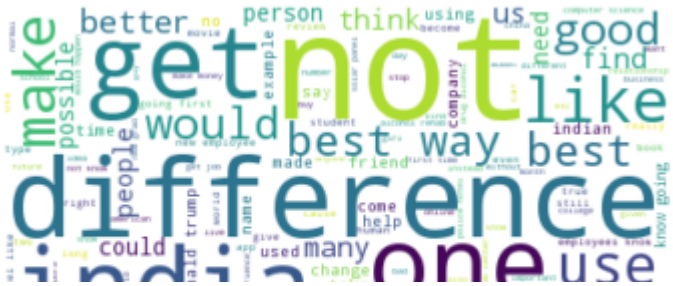
```
Collecting wordcloud
  Downloading https://files.pythonhosted.org/packages/ae/af/849edf14d573eba9c8082db8
    100% |████████████████████████████████| 368kB 7.1MB/s
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (fr
Installing collected packages: wordcloud
Successfully installed wordcloud-1.5.0
```

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

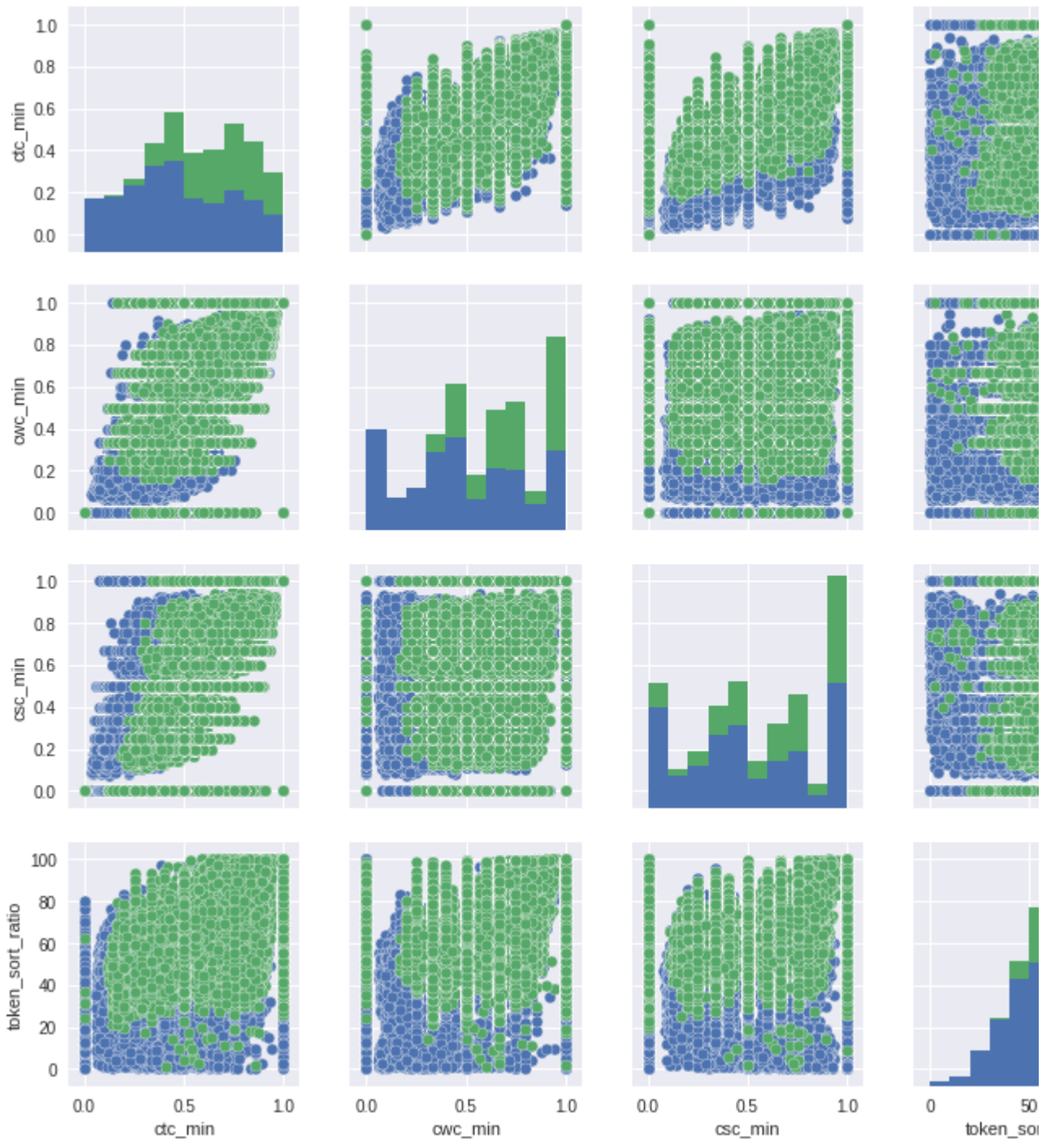
```
Word Cloud for Duplicate Question pairs
```


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

Word Cloud for non-Duplicate Question pairs:

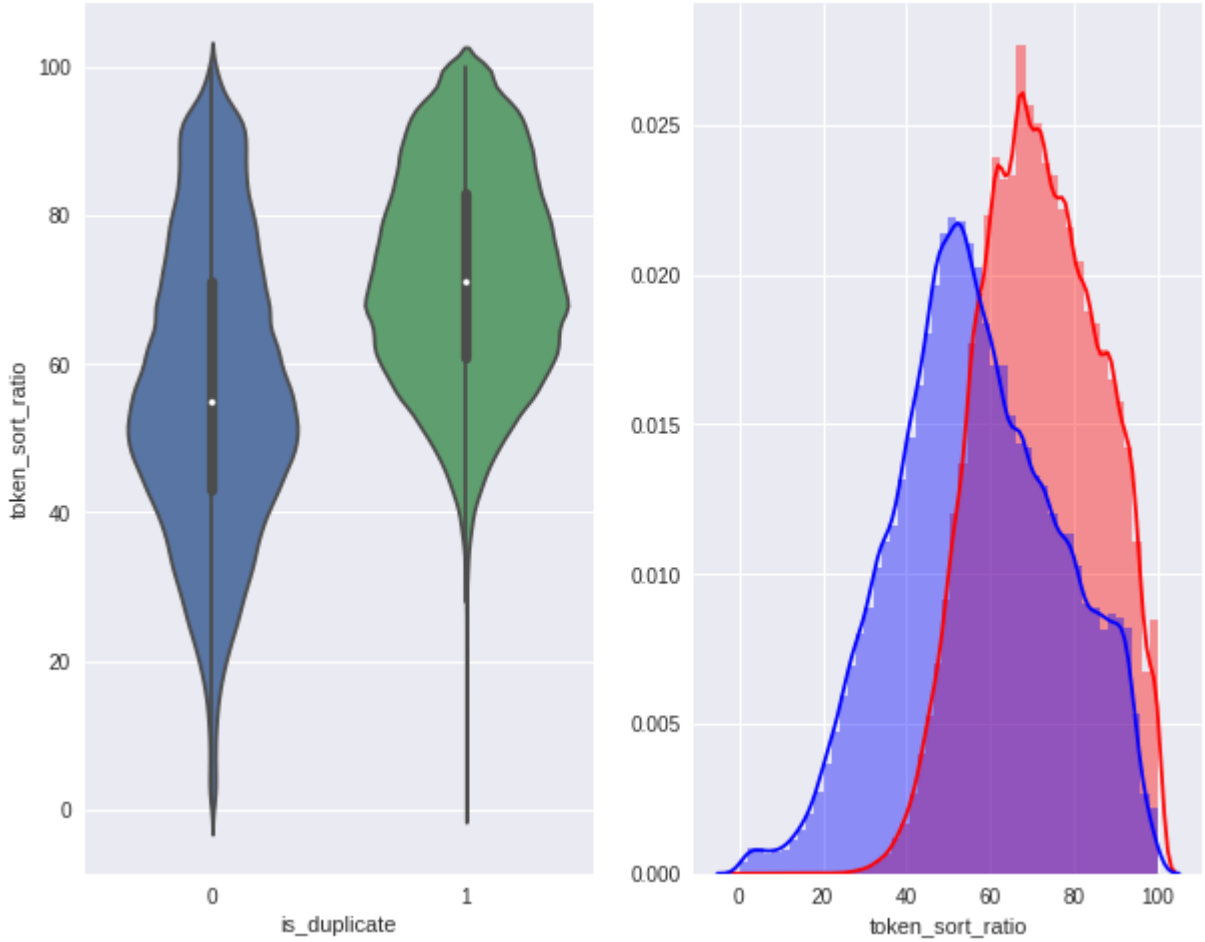



```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n]
plt.show()
```



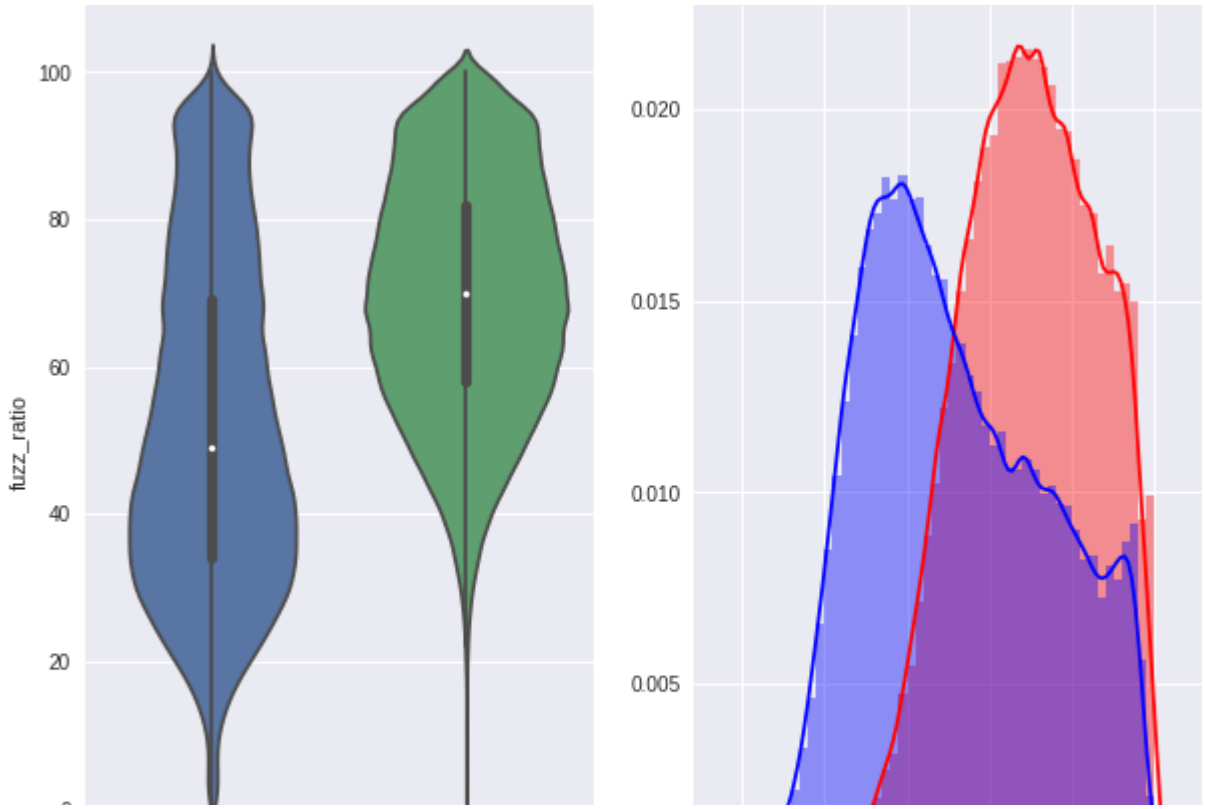
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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 = '
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" , color =
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: re
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: re
  violin_data = remove_na(group_data)
```




```
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: re
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: re
  violin_data = remove_na(group_data)
```

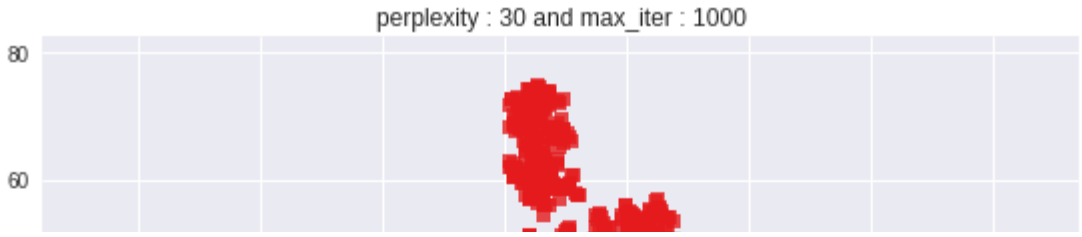



```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_max'
y = dfp_subsampled['is_duplicate'].values
```

```
from sklearn.manifold import TSNE
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 0.013s...
[t-SNE] Computed neighbors for 5000 samples in 0.372s...
[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 0.328s
[t-SNE] Iteration 50: error = 81.2897949, gradient norm = 0.0455700 (50 iterations i
[t-SNE] Iteration 100: error = 70.6164398, gradient norm = 0.0095177 (50 iterations
[t-SNE] Iteration 150: error = 68.9172134, gradient norm = 0.0056736 (50 iterations
[t-SNE] Iteration 200: error = 68.1004639, gradient norm = 0.0049672 (50 iterations
[t-SNE] Iteration 250: error = 67.5914536, gradient norm = 0.0039700 (50 iterations
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.591454
[t-SNE] Iteration 300: error = 1.7926962, gradient norm = 0.0011878 (50 iterations i
[t-SNE] Iteration 350: error = 1.3936826, gradient norm = 0.0004807 (50 iterations i
[t-SNE] Iteration 400: error = 1.2281071, gradient norm = 0.0002778 (50 iterations i
[t-SNE] Iteration 450: error = 1.1385784, gradient norm = 0.0001864 (50 iterations i
[t-SNE] Iteration 500: error = 1.0835493, gradient norm = 0.0001437 (50 iterations i
[t-SNE] Iteration 550: error = 1.0471643, gradient norm = 0.0001152 (50 iterations i
```

```
df55 = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})
# draw the plot in appropriate place in the grid
sns.lmplot(data=df55, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",marke
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```




```
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 0.015s...
[t-SNE] Computed neighbors for 5000 samples in 0.375s...
[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 0.331s
[t-SNE] Iteration 50: error = 80.5298615, gradient norm = 0.0306586 (50 iterations i
[t-SNE] Iteration 100: error = 69.3777008, gradient norm = 0.0037944 (50 iterations
[t-SNE] Iteration 150: error = 67.9726028, gradient norm = 0.0017517 (50 iterations
[t-SNE] Iteration 200: error = 67.4098892, gradient norm = 0.0013384 (50 iterations
[t-SNE] Iteration 250: error = 67.0977859, gradient norm = 0.0009594 (50 iterations
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.097786
[t-SNE] Iteration 300: error = 1.5276405, gradient norm = 0.0007237 (50 iterations i
[t-SNE] Iteration 350: error = 1.1820400, gradient norm = 0.0002119 (50 iterations i
[t-SNE] Iteration 400: error = 1.0407882, gradient norm = 0.0001023 (50 iterations i
[t-SNE] Iteration 450: error = 0.9688321, gradient norm = 0.0000652 (50 iterations i
[t-SNE] Iteration 500: error = 0.9303923, gradient norm = 0.0000554 (50 iterations i
[t-SNE] Iteration 550: error = 0.9110239, gradient norm = 0.0000524 (50 iterations i
[t-SNE] Iteration 600: error = 0.9016075, gradient norm = 0.0000421 (50 iterations i
[t-SNE] Iteration 650: error = 0.8924681, gradient norm = 0.0000360 (50 iterations i
[t-SNE] Iteration 700: error = 0.8837291, gradient norm = 0.0000353 (50 iterations i
[t-SNE] Iteration 750: error = 0.8771634, gradient norm = 0.0000316 (50 iterations i
[t-SNE] Iteration 800: error = 0.8718039, gradient norm = 0.0000295 (50 iterations i
[t-SNE] Iteration 850: error = 0.8669323, gradient norm = 0.0000276 (50 iterations i
[t-SNE] Iteration 900: error = 0.8628623, gradient norm = 0.0000262 (50 iterations i
[t-SNE] Iteration 950: error = 0.8591092, gradient norm = 0.0000241 (50 iterations i
[t-SNE] Iteration 1000: error = 0.8553245, gradient norm = 0.0000220 (50 iterations
[t-SNE] Error after 1000 iterations: 0.855325
```

```
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# avoid decoding problems
# 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))
```

```
df.head()
```

	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	Find the remainder when	0

```
questions = list(df['question1']) + list(df['question2'])
```

```
len(questions)
```

```
808580
```

```
tfidf = TfidfVectorizer(lowercase=False)
tfidfvec=tfidf.fit_transform(questions)
tfidffeat = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

```
import os
from os import path
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  notebook again ")

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 notebook again"
```

```
dfnlp.head()
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len
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
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
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
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

```
dfppro.head()
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len
0	0	1	2	What is the step by step guide to invest in	What is the step by step guide to invest in sh...	0	1	1	66

```
df.head()
```

	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	Find the remainder when	0

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

```
df1.head()
```

	id	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	wor
0	0	0	1	1	66	57	14	12	
1	1	0	4	1	51	88	8	13	
2	2	0	1	1	73	59	14	10	
3	3	0	1	1	50	65	11	9	
4	4	0	3	1	76	39	13	7	

5 rows × 28 columns

```
df2.head()
```

```
   id  freq qid1  freq qid2  q1len  q2len  q1 n words  q2 n words  word Common  word
```

```
df3.head()
```

	id
0	0
1	1
2	2
3	3
4	4

```
vec1=tfidf.transform(df['question1'])
vec2=tfidf.transform(df['question2'])
```

```
vec1.shape
```

```
(404290, 86516)
```

```
vec2.shape
```

```
(404290, 86516)
```

```
import scipy.sparse as sp
vec = sp.hstack((vec1,vec2))
```

```
vec.shape
```

```
(404290, 173032)
```

```
import scipy
tt1=scipy.sparse.csr_matrix(df1.values)
```

```
tt1.shape
```

```
(404290, 28)
```

```
tt2=scipy.sparse.csr_matrix(df2.values)
```

```
tt2.shape
```

```
(404290, 12)
```

```
vec3=sp.hstack((tt1,tt2))
```

```
vec3.shape
```

```
(404290, 40)
```

```
final=sp.hstack((vec3,vec))
```

```
final.shape
```

```
(404290, 173072)
```

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

```
y_true.shape
```

```
(404290,)
```

```
! pip install mlxtend
```

```
Collecting mlxtend
  Downloading https://files.pythonhosted.org/packages/d0/f9/798cb32550dcbc9e0e3c143d
    100% |████████████████████████████████| 1.3MB 5.3MB/s
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: pandas>=0.17.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: matplotlib>=1.5.1 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: python-dateutil>=2 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/loca
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-package
Installing collected packages: mlxtend
Successfully installed mlxtend-0.13.0
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation 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.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss

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

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


x_train,x_test, y_train, y_test = train_test_split(final, y_true, stratify=y_true, test_siz


print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(283003, 173072)
(121287, 173072)
(283003,)
(121287,)
```

```
y_test.value_counts()
```

```
0    76508
1    44779
Name: is_duplicate, dtype: int64
```

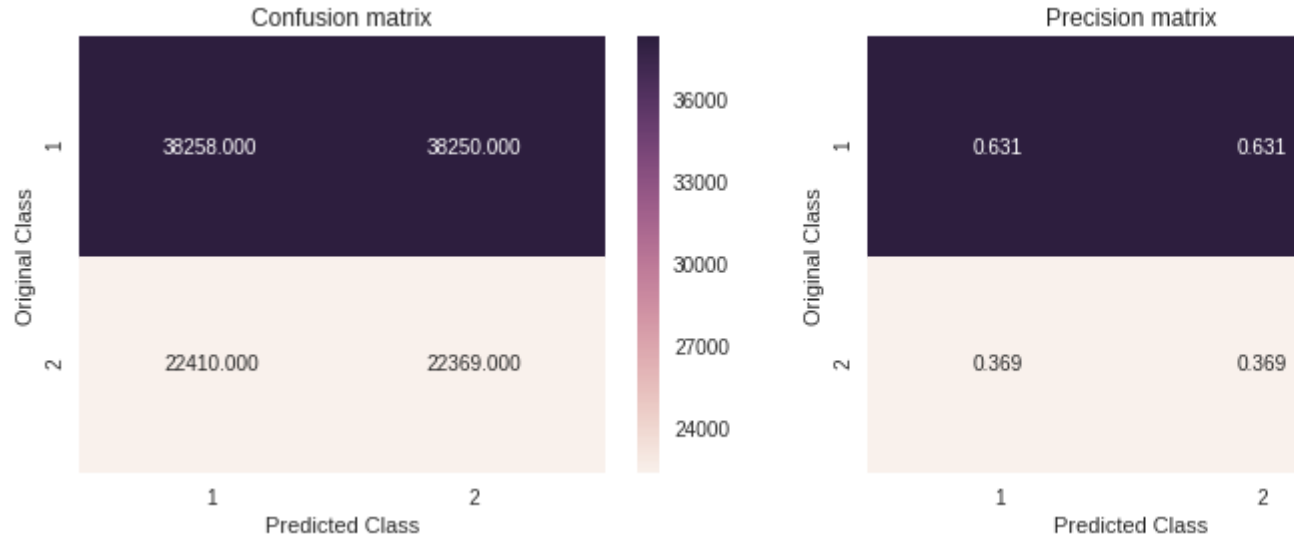
```
import seaborn as sns


# 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

    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 d
    # 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 d
    # 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, 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, 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, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

```
train_len = len(y_train)
test_len = len(y_test)
```

```
# 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)
```

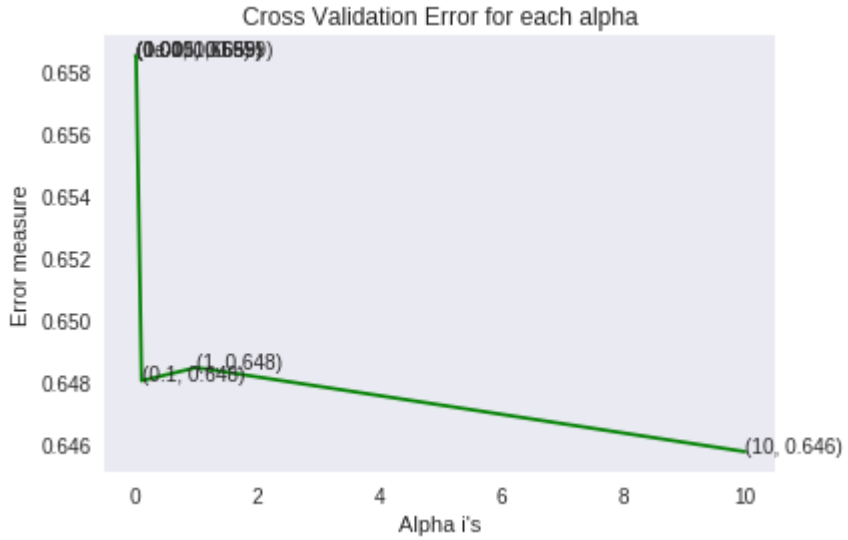
```
Log loss on Test Data using Random Model 0.8870260715318586
```




```
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/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# 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 Desce
# predict(X)  Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    log = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    log.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(log, 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=log.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, label

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)
log = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
log.fit(x_train, y_train)
sig_clf = CalibratedClassifierCV(log, 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
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_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
For values of alpha =  10 The log loss is: 0.6457434109115058
```




```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:435: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:445: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:447: RuntimeWarning: i
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:435: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
For values of best alpha =  10 The train log loss is: 0.6457742702605129
```

```
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/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# 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 Desce
# predict(X)  Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    svm = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    svm.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(svm, 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=svm.classes_, eps=1e-15))
        print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, label

    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)
    svm = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
    svm.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(svm, 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
    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_
    predicted_y =np.argmax(predict_y,axis=1)
    print("Total number of data points :", len(predicted_y))
    plot_confusion_matrix(y_test, predicted_y)
```


```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:435: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:445: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:447: RuntimeWarning: i
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:435: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:1
  "and default tol will be 1e-3." % type(self), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:435: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:445: RuntimeWarning: o
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:447: RuntimeWarning: i
  TEP_minus_T1P = P * (T * E - T1)
```

```
model = xgb.XGBClassifier()
param_dist = {"max_depth": [3,4,5,6],
              "learning_rate": [0.78,0.2,0.3,0.1,0.02],}
grid_search = GridSearchCV(model, param_grid=param_dist)
grid_search.fit(x_train, y_train)

grid_search.best_estimator_
```

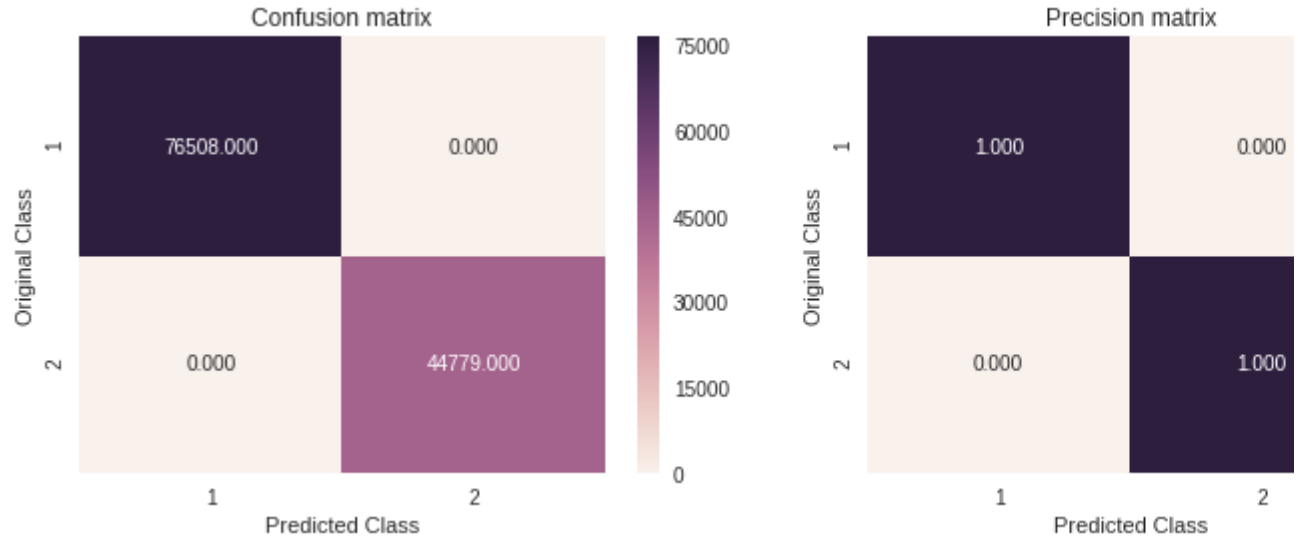
```
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:151: Deprecati
  if diff:
```

```python
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.78
params['max_depth'] = 3

d_train = xgb.DMatrix(x_train, label=y_train)
d_test = xgb.DMatrix(x_test, label=y_test)

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

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

xgdmat = xgb.DMatrix(x_train,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
[0]	train-logloss:0.19074	valid-logloss:0.19074
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopp

Will train until valid-logloss hasn't improved in 20 rounds.
[10]	train-logloss:7e-05	valid-logloss:7e-05
```

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

```
Total number of data points : 121287
```




```
from IPython.display import HTML
s = """<table>
<tr>
<th>algorithm</th>
<th>train logloss</th>
<th>test logloss</th>
</tr>
<tr>
<td>Random model</td>
<td>0.8870260715318586</td>
<td>0.8870260715318586</td>
</tr>
<tr>
<td>logistic regression</td>
<td>0.6457742702605129</td>
<td>0.6457434109115058</td>
</tr>
<tr>
<td>svm</td>
<td>0.6026941668771714</td>
<td>0.6026931927212844</td>
</tr>
<tr>
<td>gbdt</td>
<td>6e-06</td>
<td>6.408780258314279e-06</td>
</tr>

</table>"""


print("quora similarity question")
h = HTML(s)
display(h)
```

```
quora similarity question
```

algorithm	train logloss	test logloss
Random model	0.8870260715318586	0.8870260715318586
logistic regression	0.6457742702605129	0.6457434109115058
svm	0.6026941668771714	0.6026931927212844
gbdt	6e-06	6.408780258314279e-06