In [1]:

```
!curl --header "Host: doc-00-9o-docs.googleusercontent.com" --header "User-Agent: Mozilla/5
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Spee
d
100 5687M    0 5687M    0     0   105M      0 --:--:--  0:00:53 --:--:--  13
6M
```

In [2]:

```
!curl --header "Host: doc-08-9o-docs.googleusercontent.com" --header "User-Agent: Mozilla/5
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Spee
d
100  114M    0  114M    0     0  90.2M      0 --:--:--  0:00:01 --:--:-- 90.
1M
```

In [3]:

```
!curl --header "Host: doc-08-9o-docs.googleusercontent.com" --header "User-Agent: Mozilla/5
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Spee
d
100 60.4M    0 60.4M    0     0  60.6M      0 --:--:-- --:--:-- --:--:-- 60.
5M
```

In [4]:

```
!curl --header "Host: doc-0c-9o-docs.googleusercontent.com" --header "User-Agent: Mozilla/5
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Spee
d
100 5301M    0 5301M    0     0   117M      0 --:--:--  0:00:45 --:--:--  13
5M
```

In [5]:

```
!curl --header "Host: doc-10-9o-docs.googleusercontent.com" --header "User-Agent: Mozilla/5
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Spee
d
100 15.3M    0 15.3M    0     0  16.8M      0 --:--:-- --:--:-- --:--:-- 16.
8M
```

In [6]:

```
!curl --header "Host: doc-14-9o-docs.googleusercontent.com" --header "User-Agent: Mozilla/5
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Spee
d
100 31.6M    0 31.6M    0     0  47.5M      0 --:--:-- --:--:-- --:--:-- 47.
5M
```

Task

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

Quora-1.png

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

1.2 Sources/Useful Links

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

Useful Links

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in indi
a?","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 happ
en 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 geo
logist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my
Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

In [7]:

```
!pip3 install distance
!pip3 install fuzzywuzzy
```

```
Requirement already satisfied: distance in /usr/local/lib/python3.6/dist-pac
kages (0.1.3)
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.6/dist-p
ackages (0.17.0)
```

In [8]:

```
#it will ignore warning messages
import warnings
warnings.filterwarnings('ignore')
#importing numpy library
import numpy as np
#importing pandas library
import pandas as pd
#importing seaborn library
import seaborn as sns
#imorting matplotlib library
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
#importing plotly tools for 3-D
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc
#importing regular expressions
import re
from nltk.corpus import stopwords
import distance
# for data cleaning
from nltk.stem import PorterStemmer
from datetime import datetime
#importing beautiful soup for data cleaning
from bs4 import BeautifulSoup
from subprocess import check_output
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
import distance
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
import datetime as dt
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
```

```
from sklearn.svm import SVC
from sklearn.model_selection  import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

In [0]:

```
%matplotlib inline
sns.set_context('notebook')
%config InlineBackend.figure_format = 'retina'
```

3.1 Reading data and basic stats

In [10]:

```
#Reading csv file
df = pd.read_csv("train.csv")
print("Number of data points:",df.shape[0])
```

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

In [11]:

```
#printing first five rows in dataframe
df.head()
```

Out[11]:

	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 [12]:

```
# info() function to print full summary of the dataframe
df.info()
```

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

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

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

3.2.1 Distribution of data points among output classes

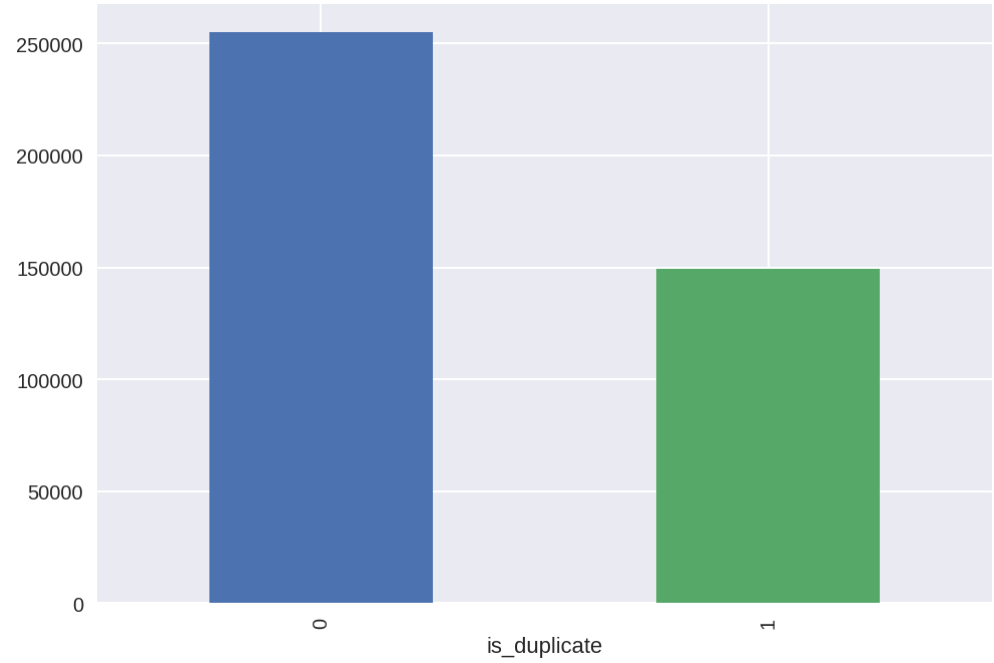
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [13]:

```
#grouping duplicate column how many points belongs to 0 or 1
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46e4368fd0>
```



In [14]:

```
#Knowing total no of question pairs in dataframe
print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

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

In [15]:

```
#knowing question which are unique in %
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round(df
#Knowing questions which are duplicate in %
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

In [16]:

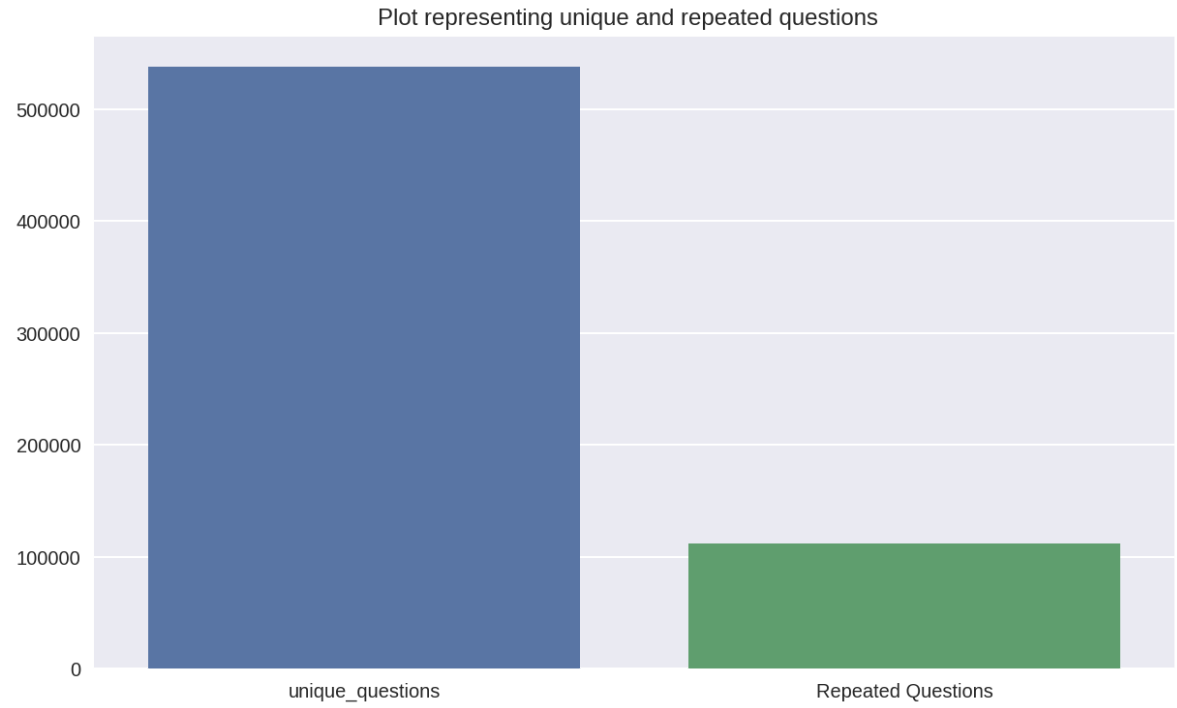
```
#joining two columns by pandas as list
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
# knowing length of ids
unique_qs = len(np.unique(qids))
#summing qids which occurs more than once
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
#printing length of questions which occurs only once
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))
#printing length questions that occurs more than once
print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(qs_mo
#printing length of questions that are repeated (how man times it is repeated)
print ('Max number of times a single question is repeated: {}\n'.format(max(qids.value_cour
#counts no of items present each individual
q_vals=qids.value_counts()
#placing values in variable
q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.779539
45937505%)

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

3.2.3 Plot representing unique and repeated questions

In [17]:

```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]
#plotting figure and assigning size
plt.figure(figsize=(10, 6))
#Placing title name
plt.title ("Plot representing unique and repeated questions  ")
#introducing bar plot for duplicate and non duplicate(1)
sns.barplot(x,y)
#showing plot
plt.show()
```



3.2.4 Checking for Duplicates

In [18]:

```
#checking whether there are any repeated pair of questions
pair_duplicates = df[['qid1','qid2','is_duplicate']].groupby(['qid1','qid2']).count().reset
#printing no of questions that are duplicates
print ("Number of duplicate questions",(pair_duplicates).shape[0] - df.shape[0])
```

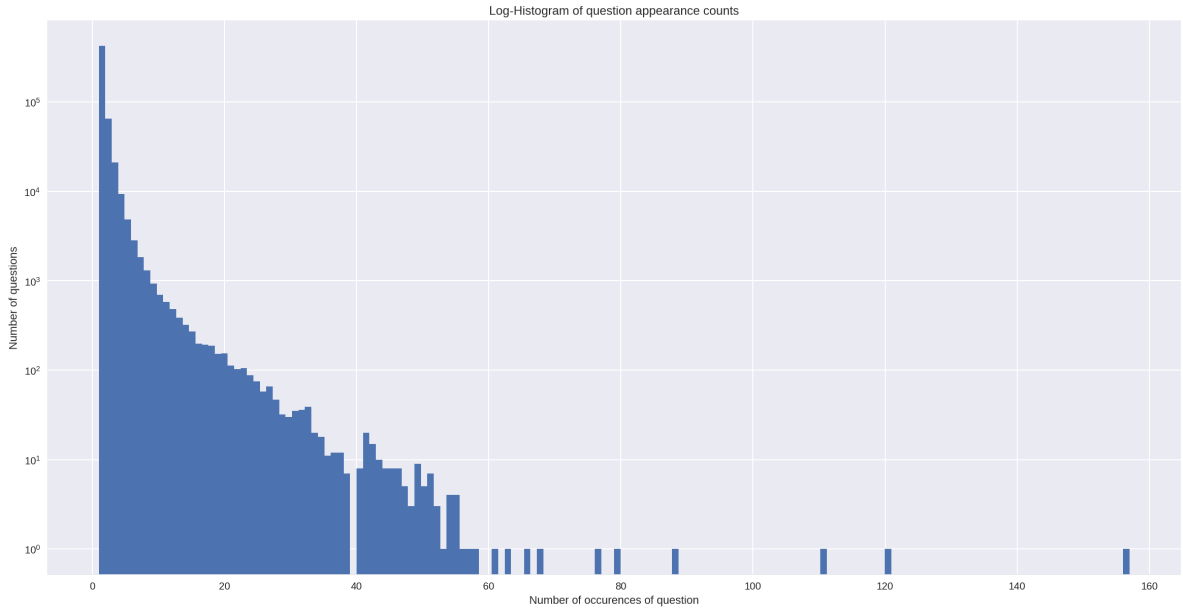
```
Number of duplicate questions 0
```

3.2.5 Number of occurrences of each question

In [19]:

```
#plotting figure with required dimensions
plt.figure(figsize=(20, 10))
#plotting histograms of qids
plt.hist(qids.value_counts(), bins=160)
#taking logscale in y-axis
plt.yscale('log', nonposy='clip')
#placing title for the figure
plt.title('Log-Histogram of question appearance counts')
#giving name to xlabel
plt.xlabel('Number of occurences of question')
#giving name to y label
plt.ylabel('Number of questions')
#printing maximum no of questions repeated once
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
```



This plot indicates that it contains outliers 100,120 on x-axis are outliers

3.2.6 Checking for NULL values

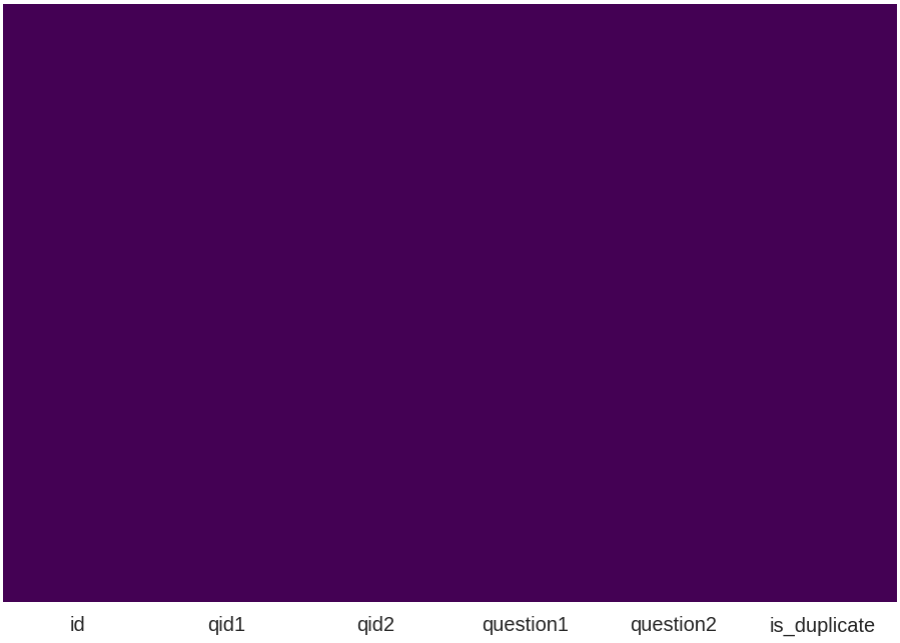
In [20]:

```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
#printing nan values present in dataframe
print (nan_rows)
sns.heatmap(df.isnull(),cbar=False,yticklabels=False,cmap = 'viridis')
```

```
           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
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46b8cd2ef0>
```



- There are two rows with null values in question2

Dataset has two missing values. That's why you would've noticed in figure represented by different colour shade on purple background. Do try it out with other dataset which has no missing values,you'll see the difference.

In [21]:

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

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1
- **q2len** = Length of q2
- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2
- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)
- **word_share** = (word_common)/(word_Total)
- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2
- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

In [22]:

```
#checking file is present or not
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
#if file is present then it will read
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
#if file is not present it will create and do necessary options of given conditions
else:
    #grouping ods and producing a DataFrame with transformed values and that has the same a
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    #grouping ods and producing a DataFrame with transformed values and that has the same a
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    #knowing length and converts the specified value into a string.
    df['q1len'] = df['question1'].str.len()
    #knowing length and converts the specified value into a string.
    df['q2len'] = df['question2'].str.len()
    #splitting the row
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    #splitting the row
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))
#building function
    def normalized_word_Common(row):
        #converting characters into lower case and returns a copy of the string with both l
        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)

    #building function
    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)
    #building function
    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)
#combining two variables
    df['freq_q1+q2'] = df['freq_qid1']+df['freq_qid2']
    df['freq_q1-q2'] = abs(df['freq_qid1']-df['freq_qid2'])
#creating csv file
    df.to_csv("df_fe_without_preprocessing_train.csv", index=False)
#printing first five rows
df.head()
```

Out[22]:

	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	

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1
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 [23]:

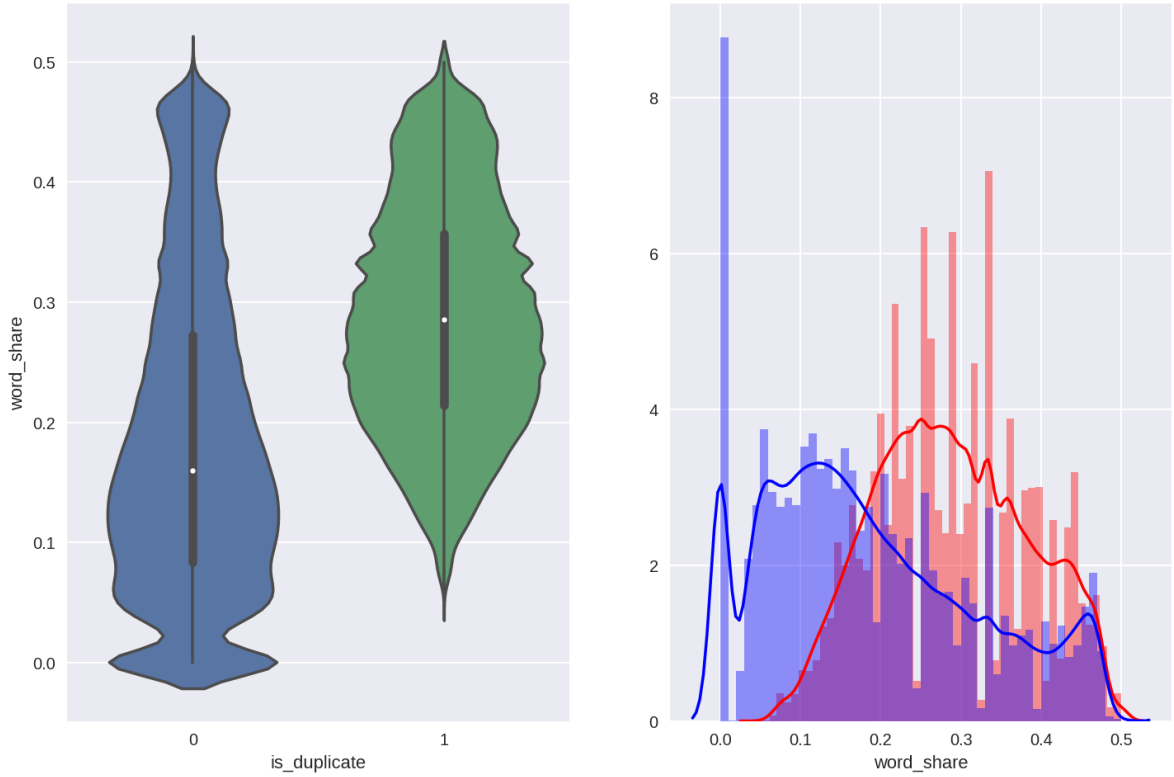
```
#printing minimum length of questions in question-1
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))
#printing minimum length of questions in question-2
print ("Minimum length of the questions in question2 : " , min(df['q2_n_words']))
#printing mo of questions that has minimum length of questions in question-1
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
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

In [24]:

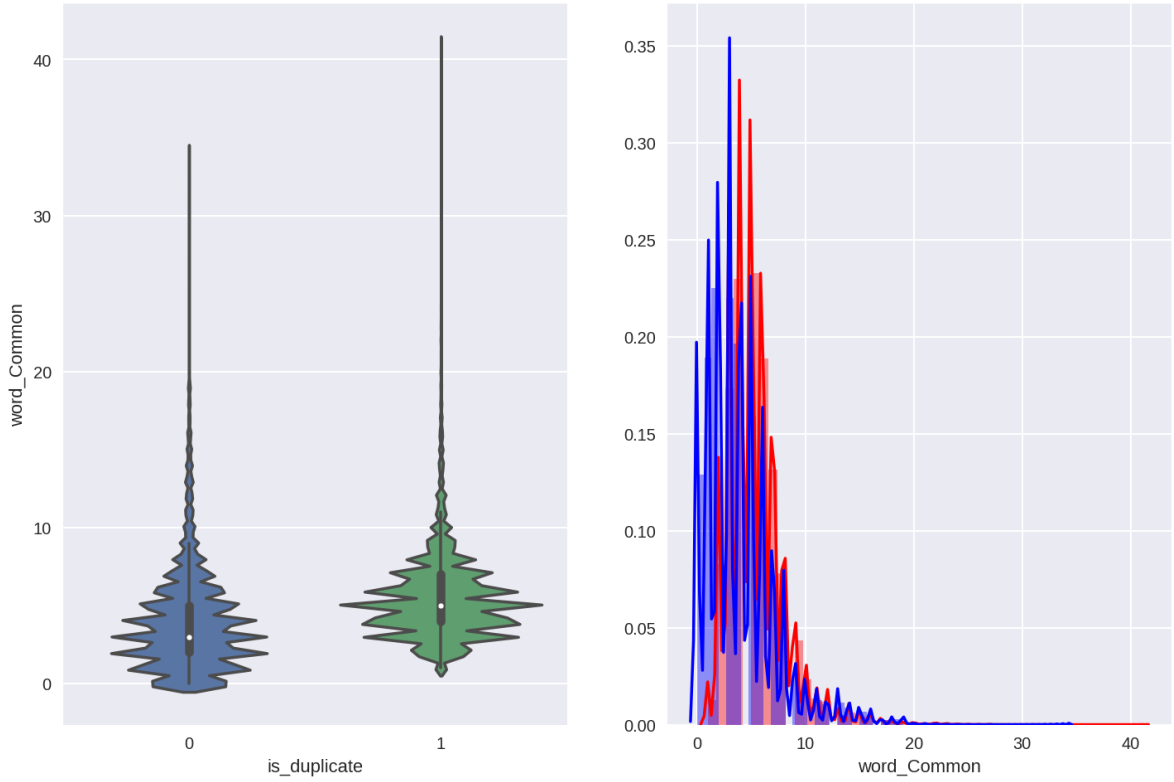
```
#plotting figure with required dimensions
plt.figure(figsize=(12, 8))
#creating subplot 1
plt.subplot(1,2,1)
#plotting violin plot
sns.violinplot(x = 'is_duplicate', y = 'word_share', data = df[0:])
#creating subplot 2
plt.subplot(1,2,2)
#plotting distribution plot of dulicate
sns.distplot(df[df['is_duplicate'] == 1.0]['word_share'][0:] , label = "1", color = 'red')
#plotting distribution plot of unique
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , color = 'blue'
#show plot
plt.show()
```



3.3.1.2 Feature: word_Common

In [25]:

```
#plotting figure with required dimensions
plt.figure(figsize=(12, 8))
#creating subplot 1
plt.subplot(1,2,1)
#plotting violin plot
sns.violinplot(x = 'is_duplicate', y = 'word_Common', data = df[0:])
#creating subplot 2
plt.subplot(1,2,2)
#plotting distribution plot of dulicate
sns.distplot(df[df['is_duplicate'] == 1.0]['word_Common'][0:] , label = "1", color = 'red')
#plotting distribution plot of unique
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label = "0" , color = 'blue
#show plot
plt.show()
```



In [0]:

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byt
#checking file is present or not
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    #Reading csv file
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    #filling empty values
    df = df.fillna('')
    #printing first five rows
    df.head()
else:
    #printing file is present
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous noteboo
```

In [27]:

```
#printing first five rows
df.head(2)
```

Out[27]:

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

In [28]:

```
# To get the results in 4 decemal points
#downloading stop words
import nltk
nltk.download('stopwords')

SAFE_DIV = 0.0001
#loading stop words
STOP_WORDS = stopwords.words("english")

#doing processing of text
def preprocess(x):
    #characters converting into lower
    x = str(x).lower()
    #replacing values
    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)

    #doing stemming
    porter = PorterStemmer()
    pattern = re.compile('\W')
    #defining type
    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 /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

- **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
  cwc_max = common_word_count / (max(len(q1_words), len(q2_words))
- **csc_min** : Ratio of common_stop_count to min length of stop count of Q1 and Q2
  csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))
- **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))
- **ctc_min** : Ratio of common_token_count to min lenghth of token count of Q1 and Q2
  ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))
- **ctc_max** : Ratio of common_token_count to max lenghth of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))
- **last_word_eq** : Check if First word of both questions is equal or not
  last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])
- **first_word_eq** : Check if First word of both questions is equal or not
  first_word_eq = int(q1_tokens[0] == q2_tokens[0])
- **abs_len_diff** : Abs. length difference
  abs_len_diff = abs(len(q1_tokens) - len(q2_tokens))
- **mean_len** : Average Token Length of both Questions
  mean_len = (len(q1_tokens) + len(q2_tokens))/2
- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)
- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)
- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)
- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)
- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [0]:

```
#Tokenizing features
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
```

In [30]:

```
#checking file is present or not
if os.path.isfile('nlp_features_train.csv'):
    #Reading csv file
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    #filling empty values
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    #creating csv file
    df.to_csv("nlp_features_train.csv", index=False)
#printing first five rows
df.head(2)
```

Out[30]:

	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

In [31]:

```
#storing values in newvariable that are duplicates
df_duplicate = df[df['is_duplicate'] == 1]
#storing values in new variable that are unique
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
```

In [32]:

```
# reading the text files and removing the Stop Words:
from os import path
import nltk
nltk.download('stopwords')
from wordcloud import WordCloud, STOPWORDS

from nltk.corpus import stopwords


d = path.dirname('.')

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

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193130
```

3.5.1.2 Word Clouds generated from duplicate pair question's text __

In [33]:

```
#generating word cloud
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
```

3.5.1.3 Word Clouds generated from non duplicate pair question's text __

In [34]:

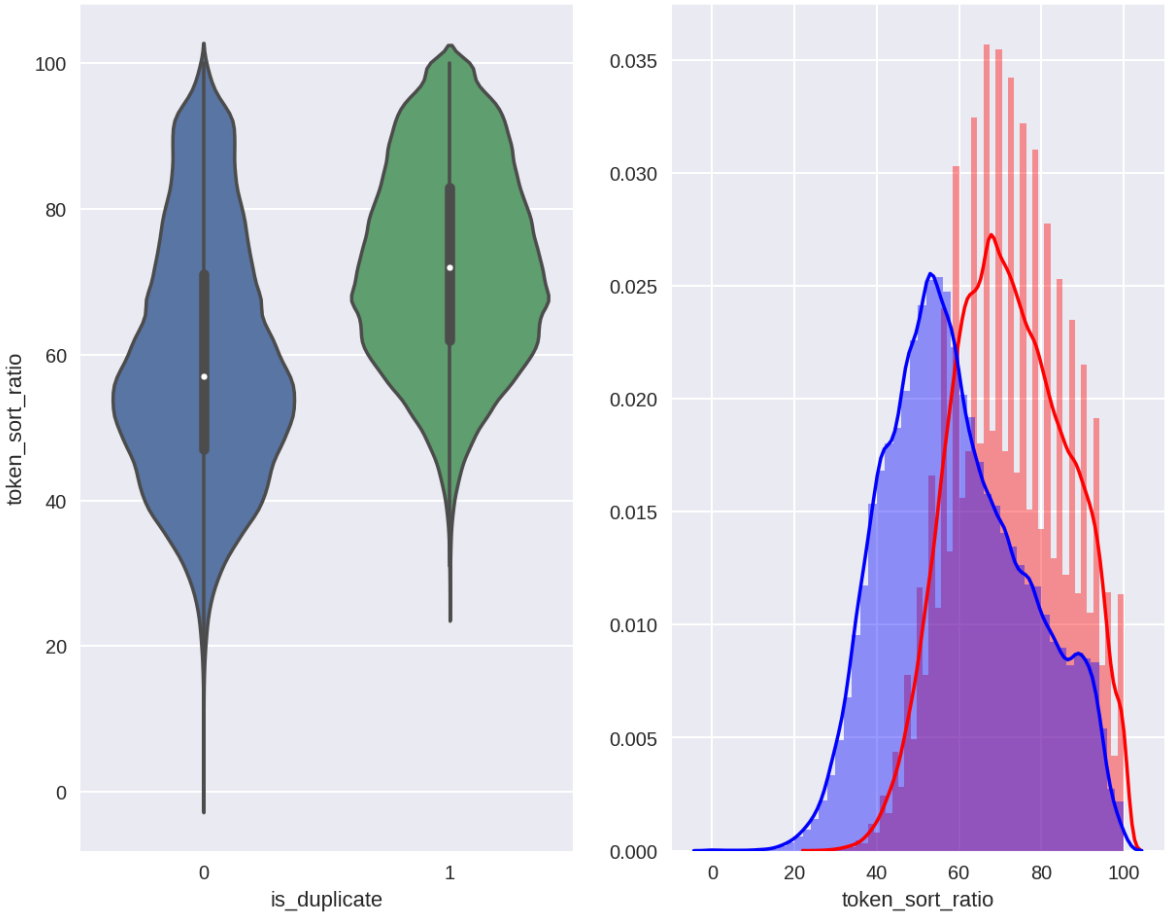
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

Feature: token_sort_ratio

In [35]:

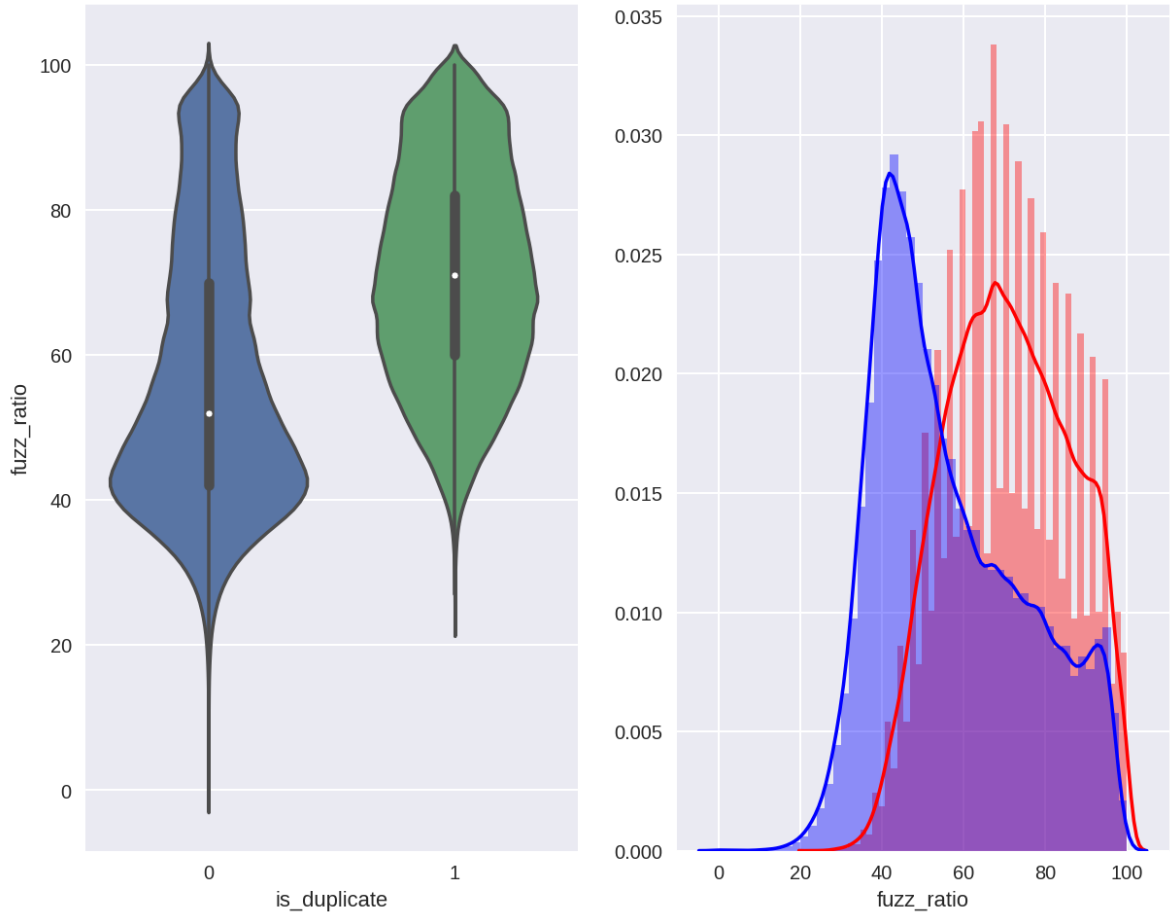
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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()
```



feature: fuzzy_ratio

In [36]:

```
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()
```



- 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)

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

Visualization

In [0]:

```
# 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
```

In [38]:

```
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 0.014s...
[t-SNE] Computed neighbors for 5000 samples in 0.412s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.309s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iter
ations in 2.596s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 ite
rations in 1.931s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 ite
rations in 1.958s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 ite
rations in 2.025s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 ite
rations in 2.024s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.27334
6
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 iter
ations in 1.959s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iter
ations in 1.893s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iter
ations in 1.904s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iter
ations in 1.931s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iter
ations in 1.931s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iter
ations in 1.970s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iter
ations in 1.952s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iter
ations in 1.957s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iter
ations in 1.960s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iter
ations in 1.960s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iter
ations in 1.953s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iter
ations in 1.959s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iter
ations in 2.013s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iter
```

```
ations in 2.007s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 ite
rations in 1.998s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [39]:

```
df1 = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# draw the plot in appropriate place in the grid
sns.lmplot(data=df1, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",marker
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```

In [40]:

```
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 0.022s...
[t-SNE] Computed neighbors for 5000 samples in 0.472s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.297s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 iter
ations in 12.617s)
[t-SNE] Iteration 100: error = 69.1127167, gradient norm = 0.0036756 (50 ite
rations in 6.388s)
[t-SNE] Iteration 150: error = 67.6178818, gradient norm = 0.0017629 (50 ite
rations in 5.620s)
[t-SNE] Iteration 200: error = 67.0571747, gradient norm = 0.0011826 (50 ite
rations in 5.590s)
[t-SNE] Iteration 250: error = 66.7298050, gradient norm = 0.0008528 (50 ite
rations in 5.969s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.72980
5
[t-SNE] Iteration 300: error = 1.4963876, gradient norm = 0.0006857 (50 iter
ations in 7.442s)
[t-SNE] Iteration 350: error = 1.1549060, gradient norm = 0.0001911 (50 iter
ations in 9.839s)
[t-SNE] Iteration 400: error = 1.0083323, gradient norm = 0.0000968 (50 iter
ations in 9.995s)
[t-SNE] Iteration 450: error = 0.9356370, gradient norm = 0.0000660 (50 iter
ations in 9.953s)
[t-SNE] Iteration 500: error = 0.8982100, gradient norm = 0.0000521 (50 iter
ations in 9.707s)
[t-SNE] Iteration 550: error = 0.8778670, gradient norm = 0.0000595 (50 iter
ations in 9.584s)
[t-SNE] Iteration 600: error = 0.8642665, gradient norm = 0.0000579 (50 iter
ations in 9.672s)
[t-SNE] Iteration 650: error = 0.8558875, gradient norm = 0.0000362 (50 iter
ations in 9.674s)
[t-SNE] Iteration 700: error = 0.8492573, gradient norm = 0.0000305 (50 iter
ations in 9.786s)
[t-SNE] Iteration 750: error = 0.8432317, gradient norm = 0.0000276 (50 iter
ations in 9.843s)
[t-SNE] Iteration 800: error = 0.8378869, gradient norm = 0.0000279 (50 iter
ations in 9.734s)
[t-SNE] Iteration 850: error = 0.8331724, gradient norm = 0.0000261 (50 iter
ations in 9.573s)
[t-SNE] Iteration 900: error = 0.8291837, gradient norm = 0.0000259 (50 iter
ations in 9.454s)
```

```
[t-SNE] Iteration 950: error = 0.8255505, gradient norm = 0.0000333 (50 iter
ations in 9.416s)
[t-SNE] Iteration 1000: error = 0.8224180, gradient norm = 0.0000235 (50 ite
rations in 9.326s)
[t-SNE] KL divergence after 1000 iterations: 0.822418
```

In [41]:

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

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

In [42]:

```
#(Q) What are the column names in our dataset?
print (df.columns)
```

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [43]:

```
#(Q) How many data points for each class are present?
df["is_duplicate"].value_counts()
# balanced-dataset vs imbalanced datasets
```

Out[43]:

```
0    255027
1    149263
Name: is_duplicate, dtype: int64
```

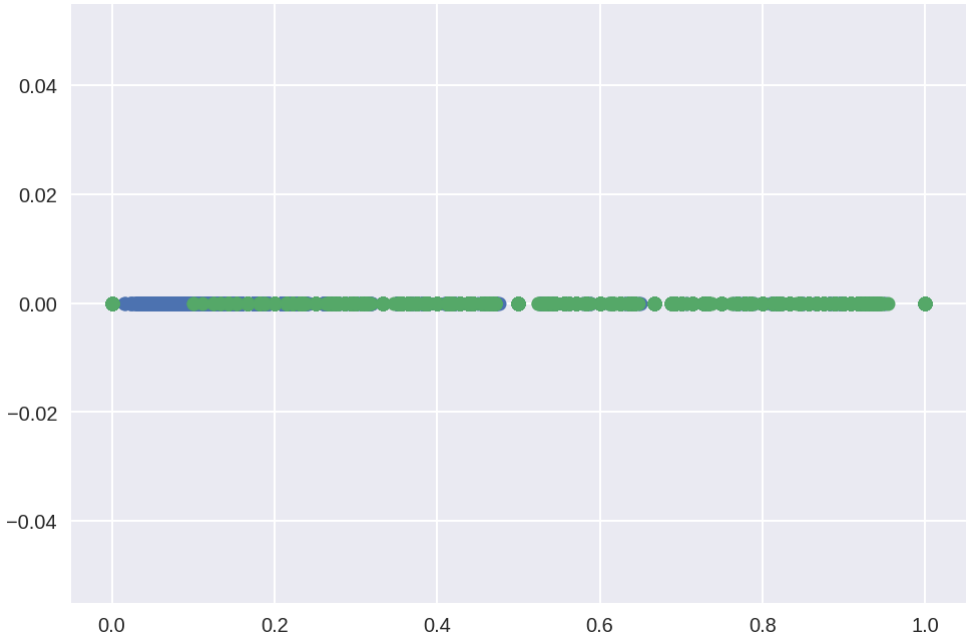
Observation(s):

1. It is Imbalancded dataset

Univariate Analysis

In [44]:

```
# What about 1-D scatter plot using just one feature?
#1-D scatter plot
import numpy as np
notduplicate = df.loc[df["is_duplicate"] == 0.0];
duplicate = df.loc[df["is_duplicate"] == 1.0];

plt.plot(notduplicate["cwc_max"], np.zeros_like(notduplicate['cwc_max']), 'o')
plt.plot(duplicate["cwc_max"], np.zeros_like(duplicate['cwc_max']), 'o')

plt.show()
#Disadvantages of 1-D scatter plot: Very hard to make sense as points
#are overlapping a lot.
#Are there better ways of visualizing 1-D scatter plots?
```



In [45]:

```
# Plots of CDF of token_sort_ratio for various types

# Misclassification error if you use token_sort_ratio only.

counts, bin_edges = np.histogram(notduplicate['cwc_max'], bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)


# duplicate
counts, bin_edges = np.histogram(duplicate['cwc_max'], bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)

plt.show();
```

```
[0.18988186 0.13271536 0.12384963 0.15329357 0.13567975 0.04741067
 0.07320009 0.08025033 0.03834104 0.02537771]
[0.         0.09999944 0.19999889 0.29999833 0.39999778 0.49999722
 0.59999667 0.69999611 0.79999556 0.899995   0.99999444]
[0.00094464 0.00463611 0.03960124 0.17138876 0.22580278 0.11565492
 0.13941834 0.15559114 0.03253988 0.11442219]
[0.         0.09999958 0.19999917 0.29999875 0.39999833 0.49999792
 0.5999975  0.69999708 0.79999667 0.89999625 0.99999583]
```

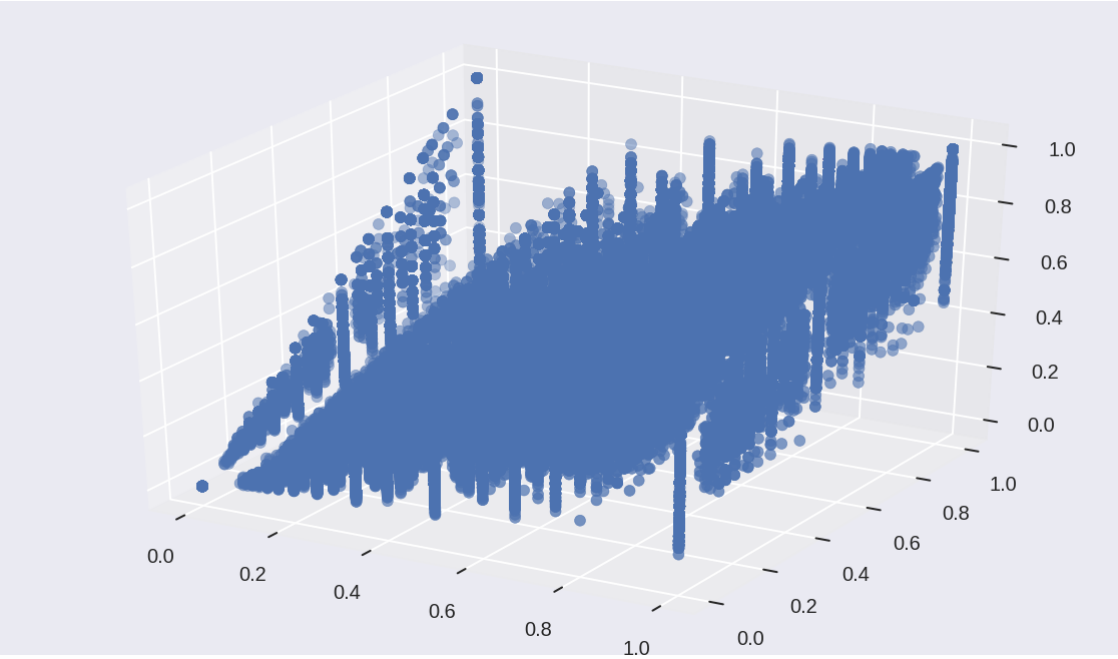
3.5.1.4 3D Scatter plot
https://plot.ly/pandas/3d-scatter-plots/ (https://plot.ly/pandas/3d-scatter-plots/)

Needs a lot to mouse interaction to interpret data.

What about 4-D, 5-D or n-D scatter plot?

In [46]:

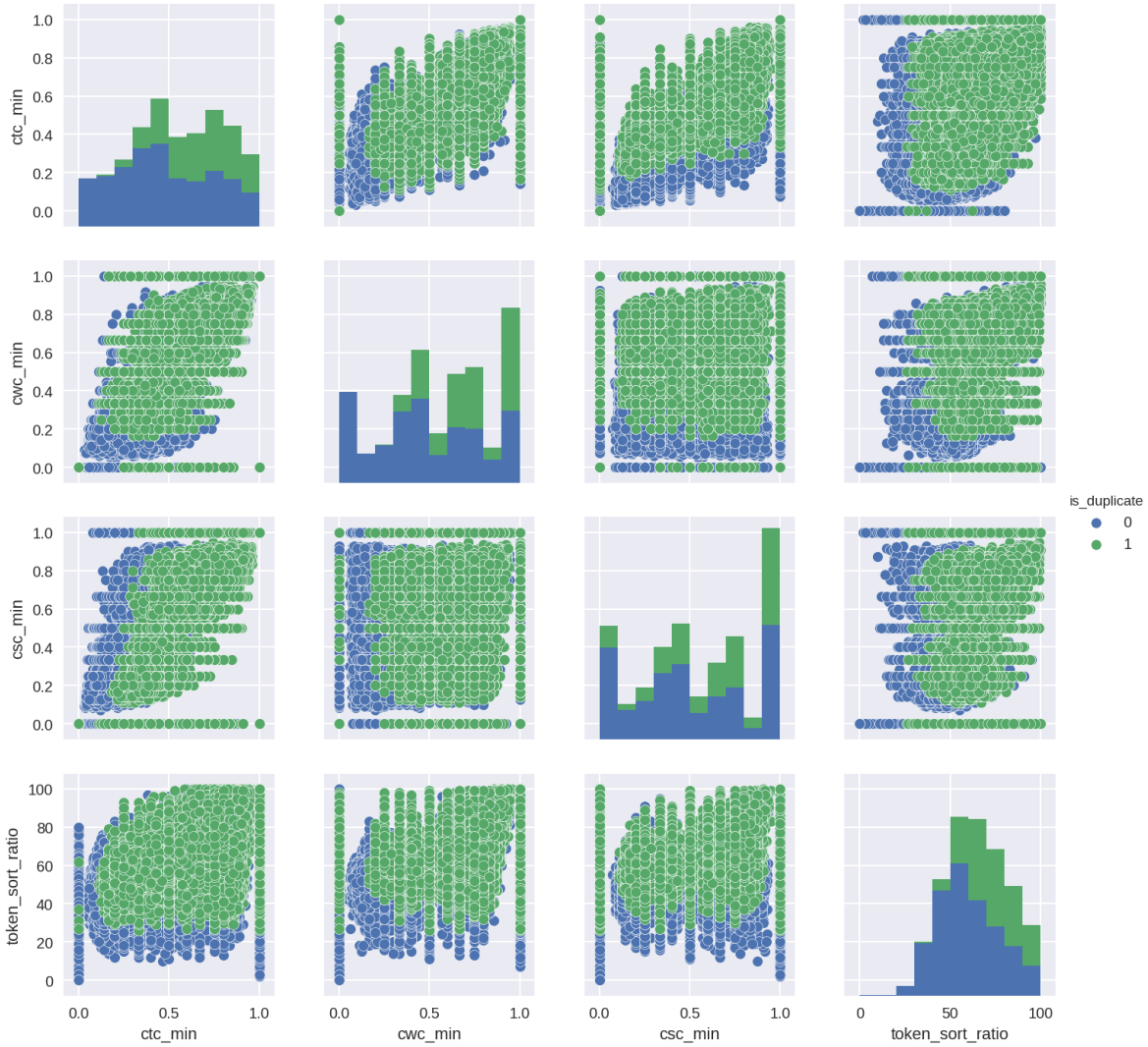
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['csc_min'],df['cwc_min'],df['ctc_min'], s=30)
plt.show()
```



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

In [47]:

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



Observation(s):

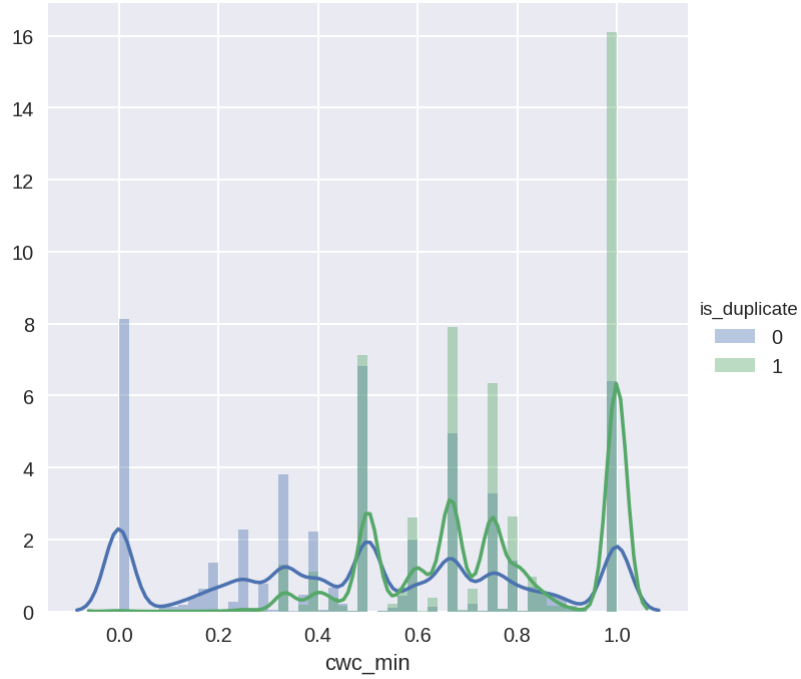
1. we can easily seperate points in ctc_min and csc_min where maximum no of points are seperated by drawing hyperplane

csc_min and cwc_min are most important features

Distribution plots

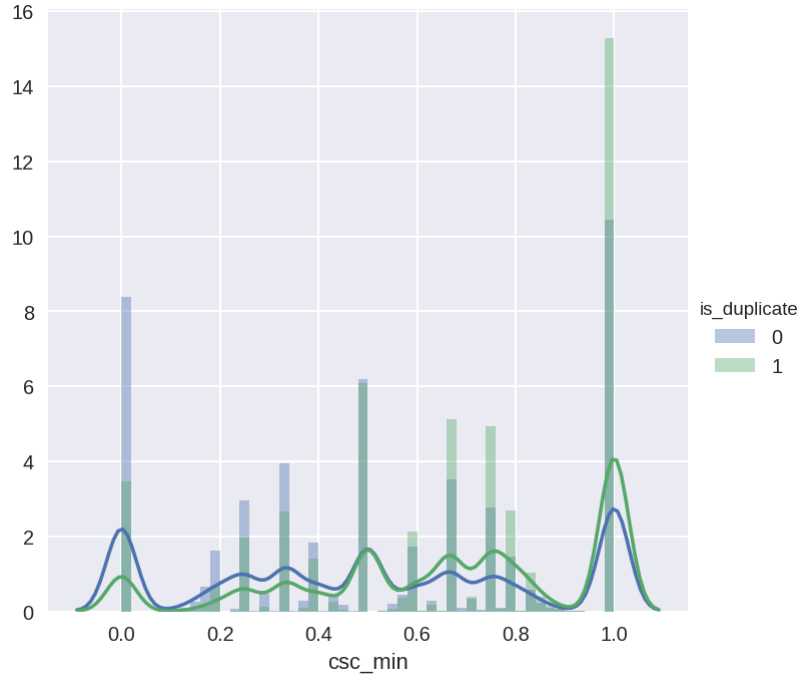
In [48]:

```
sns.FacetGrid(df, hue="is_duplicate", size=5) \
   .map(sns.distplot, "cwc_min") \
   .add_legend();
plt.show();
```



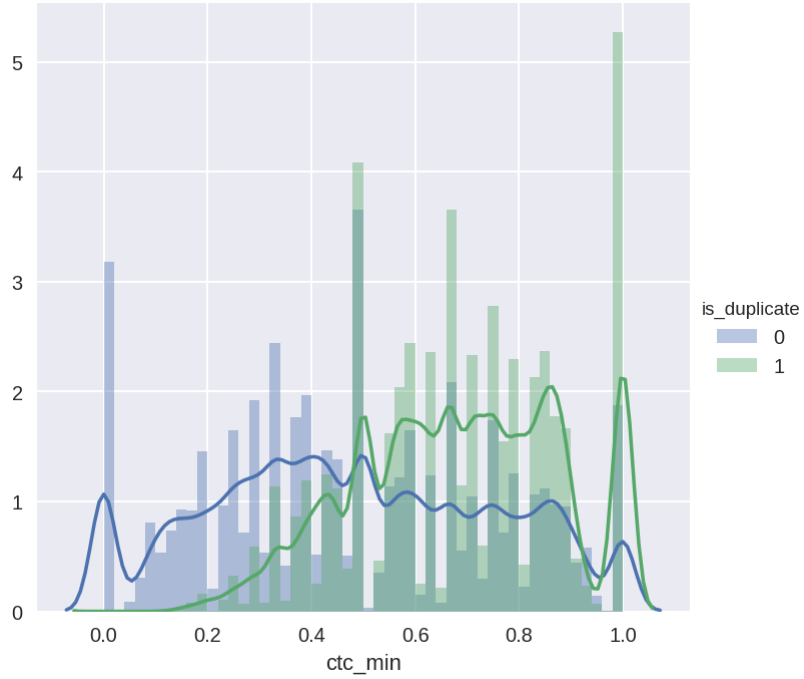
In [49]:

```
sns.FacetGrid(df, hue="is_duplicate", size=5) \
   .map(sns.distplot, "csc_min") \
   .add_legend();
plt.show();
```



In [50]:

```
sns.FacetGrid(df, hue="is_duplicate", size=5) \
   .map(sns.distplot, "ctc_min") \
   .add_legend();
plt.show();
```



Observation(s): 1.Overllapping is lot in above features so we go in detail to select top features

3.5.1.7 Mean, Variance and Std-dev

In [51]:

```
#Mean, Variance, Std-deviation,
print("\nMeans:")
print(np.mean(notduplicate["cwc_min"]))
#Mean with an outlier.
print(np.mean(np.append(notduplicate["cwc_min"],20)));
print(np.mean(duplicate["cwc_min"]))

print("\nStd-dev:");
print(np.std(notduplicate["cwc_min"]))
print(np.std(duplicate["cwc_min"]))
```

```
Means:
0.4919638139256496
0.49204030762923423
0.7501360871560521

Std-dev:
0.3197866048531545
0.20656167729713448
```

3.5.1.8 Median, Percentile, Quantile, IQR, MAD

In [52]:

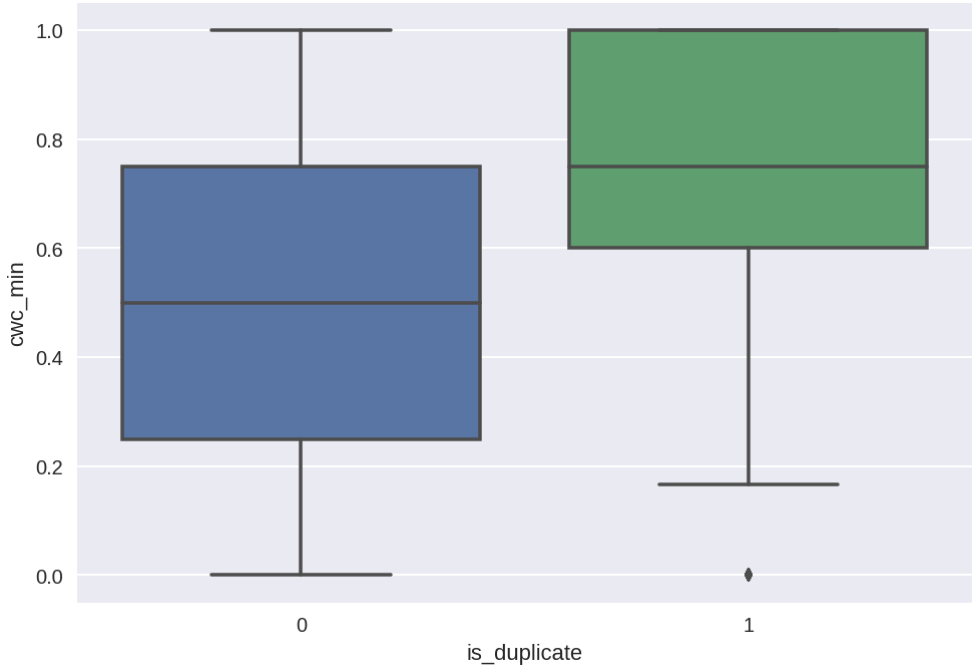
```
#Median, Quantiles, Percentiles, IQR.
print("\nMedians:")
print(np.median(notduplicate["cwc_min"]))
#Median with an outlier
print(np.median(np.append(notduplicate["cwc_min"],70)));
print(np.median(duplicate["cwc_min"]))

print("\nQuantiles:")
print(np.percentile(notduplicate["cwc_min"],np.arange(0, 100, 25)))
print(np.percentile(duplicate["cwc_min"],np.arange(0, 100, 25)))


print("\n90th Percentiles:")
print(np.percentile(notduplicate["cwc_min"],90))
print(np.percentile(duplicate["cwc_min"],90))


from statsmodels import robust
print ("\nMedian Absolute Deviation")
print(robust.mad(notduplicate["cwc_min"]))
print(robust.mad(duplicate["cwc_min"]))
```

```
Medians:
0.4999875003124922
0.4999875003124922
0.7499812504687383

Quantiles:
[0.         0.24999375 0.4999875  0.74998125]
[0.         0.599988   0.74998125 0.99995   ]

90th Percentiles:
0.9999500024998748
0.9999750006249843

Median Absolute Deviation
0.3706412885941856
0.3706042263184396
```

3.5.1.9 Boxplot

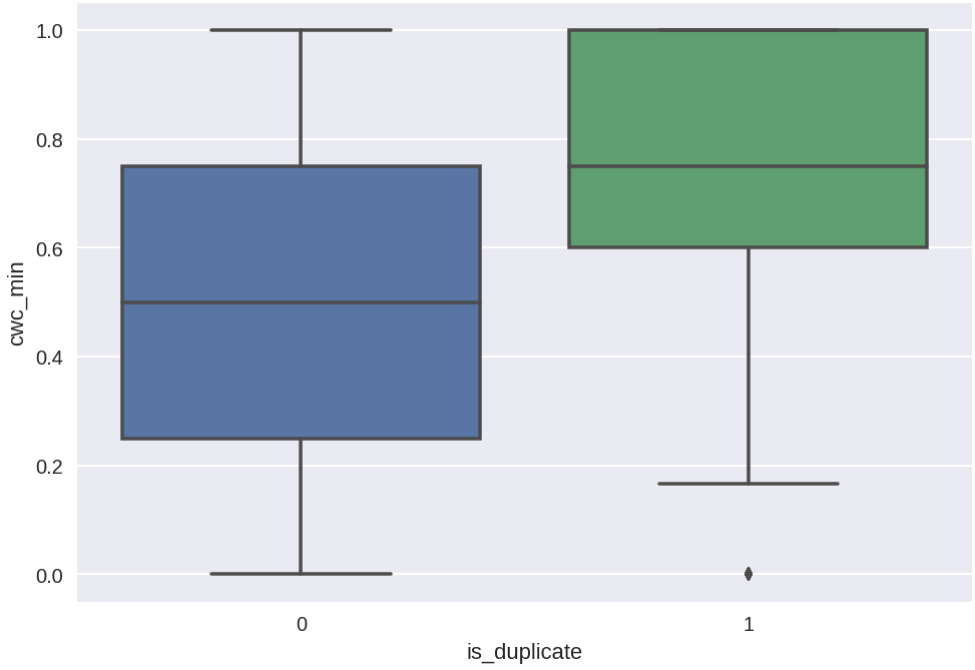
In [53]:

```
# the skewed box plot shows us the presence of outliers
sns.boxplot(x="is_duplicate",y="cwc_min", data =df)
plt.show()
```



In [54]:

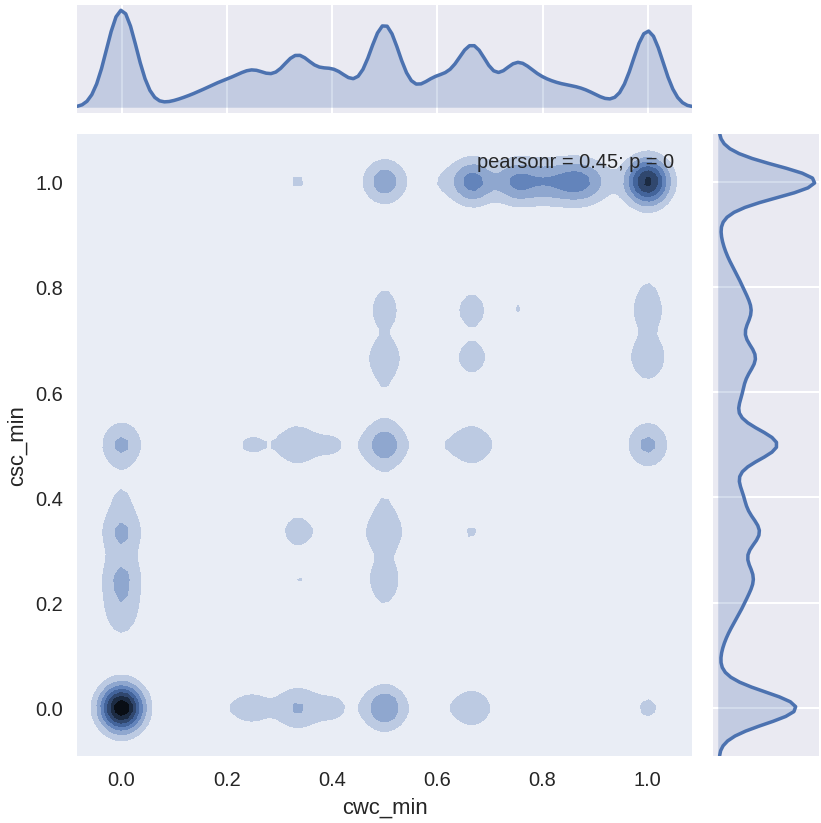
```
# the skewed box plot shows us the presence of outliers
sns.boxplot(x="is_duplicate",y="cwc_min", data =df)
plt.show()
```



3.5.1.10 Multivariate probability density, contour plot.

In [55]:

```
#2D Density plot, contors-plot
import seaborn as sns
sns.jointplot(x="cwc_min", y="csc_min", data=notduplicate, kind="kde");
plt.show();
```

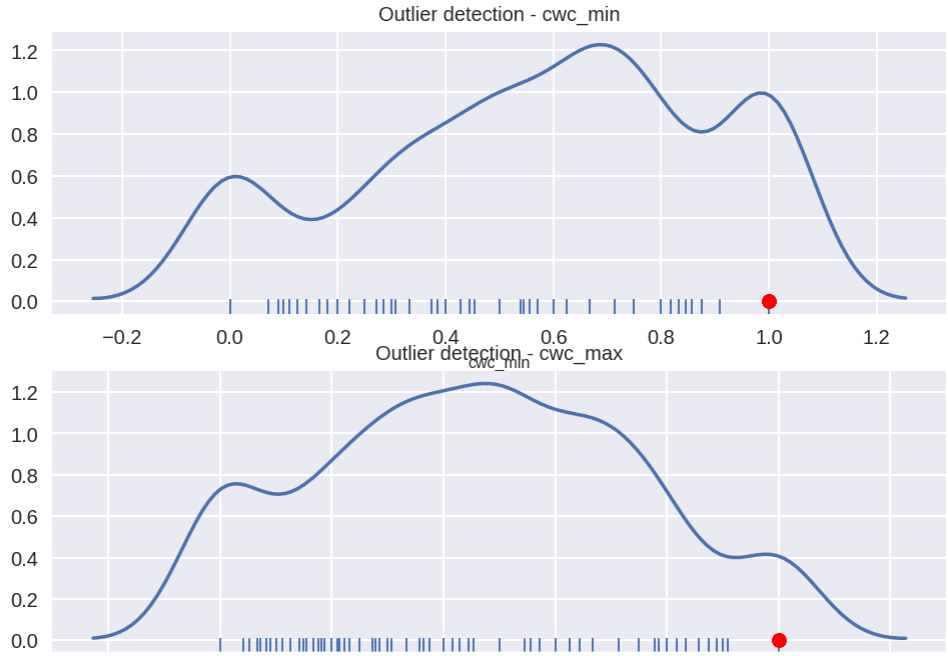


By obesrving above plots these are easily seperable

3.6 Checking outliers

In [56]:

```
def percentile_based_outlier(data, threshold=95):
    diff = (100 - threshold) / 2
    minval, maxval = np.percentile(data, [diff, 100 - diff])
    return (data < minval) | (data > maxval)

col_names = ['cwc_min','cwc_max','csc_min','csc_max','ctc_max','last_word_eq','first_word_e

fig, ax = plt.subplots(len(col_names), figsize=(8,40))

for i, col_val in enumerate(col_names):
    x = df[col_val][:1000]
    sns.distplot(x, ax=ax[i], rug=True, hist=False)
    outliers = x[percentile_based_outlier(x)]
    ax[i].plot(outliers, np.zeros_like(outliers), 'ro', clip_on=False)

    ax[i].set_title('Outlier detection - {}'.format(col_val), fontsize=10)
    ax[i].set_xlabel(col_val, fontsize=8)

plt.show()
```



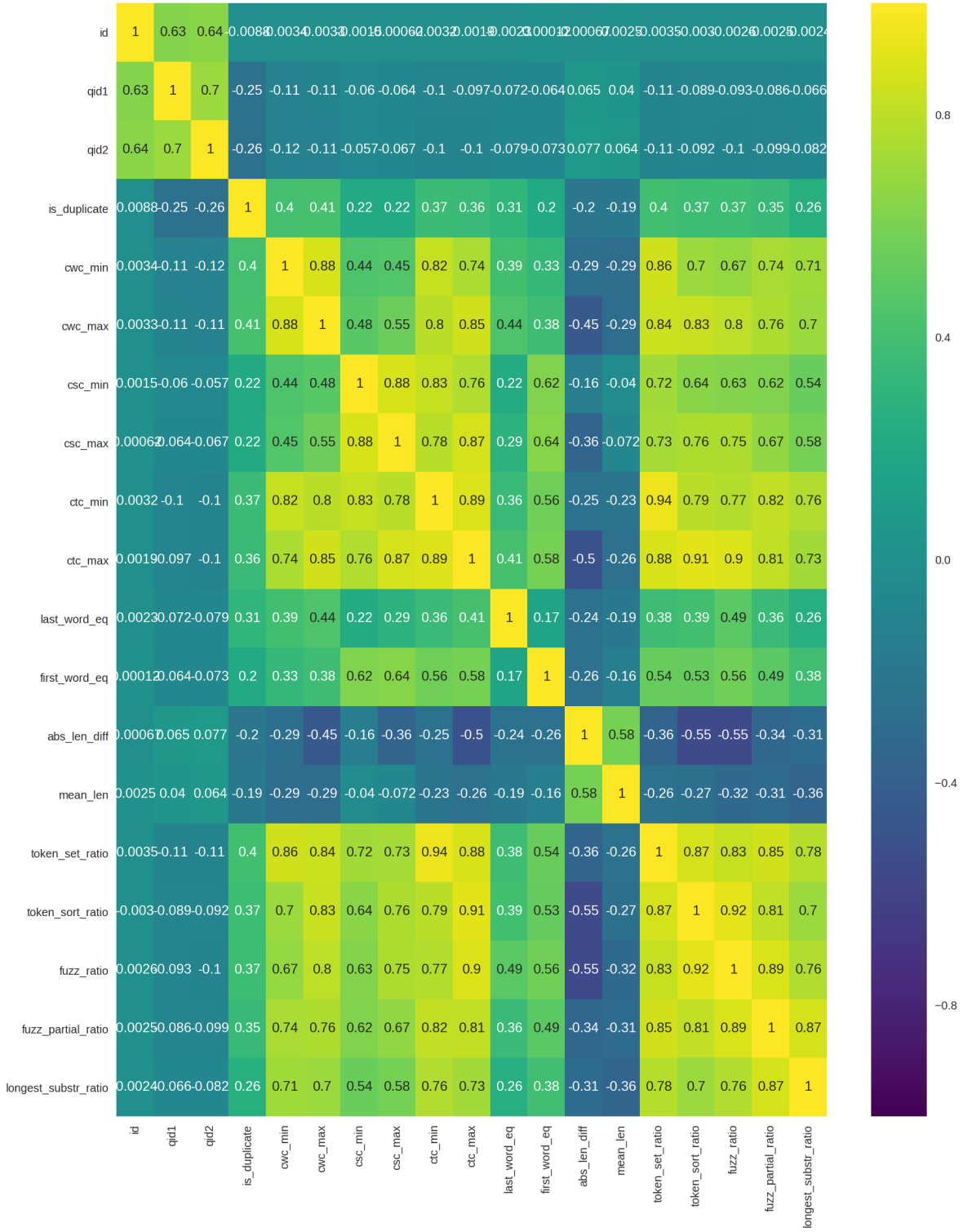
3.7 Correlation

In [57]:

```
f, ax = plt.subplots(figsize=(14,18))
corr = df.corr()
sns.heatmap(corr,xticklabels=corr.columns.values,yticklabels=corr.columns.values,annot=True

#k = 19 #number of variables for heatmap
#cols = df.corr().nlargest(k, ' is_duplicate'),df[' is_duplicate'].index
#cm = df[cols].corr()
#plt.figure(figsize=(10,6))
#sns.heatmap(cm, annot=True, cmap = 'viridis')
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46a955e080>
```



correlation value lies between -1 to +1. Highly correlated variables will have correlation value close to +1 and less correlated variables will have correlation value close to -1.The diagonal elements of the matrix value are always 1

- The most correleated values are ctc_min,token_set_ratio,longest_substr_ratio,fuzz_partial ratio

In [58]:

```
#plotting values mean median percentiles etc
print(df.describe(include='all'))
print("*"*60)
print(df.info())
```

```
                  id           qid1           qid2               question1
\
count   404290.000000  404290.000000  404290.000000                  404276
unique            NaN            NaN            NaN                  290151
top               NaN            NaN            NaN  how do i lose weight
freq              NaN            NaN            NaN                      69
mean    202144.500000  217243.942418  220955.655337                     NaN
std     116708.614502  157751.700002  159903.182629                     NaN
min          0.000000       1.000000       2.000000                     NaN
25%     101072.250000   74437.500000   74727.000000                     NaN
50%     202144.500000  192182.000000  197052.000000                     NaN
75%     303216.750000  346573.500000  354692.500000                     NaN
max     404289.000000  537932.000000  537933.000000                     NaN

                                                 question2   is_duplicate
\
count                                               404284  404290.000000
unique                                              298856            NaN
top     how can you look at someone own private instag...            NaN
```

In [59]:

```
#Read the final_features data frame
df_nlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
df_ppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
df_ppro.head(2)
```

Out[59]:

	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	

In [60]:

```
df1 = df_nlp.drop(['qid1','qid2'],axis=1)
df2 = df_ppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df = df1.merge(df2, on='id',how='left')
df.head()
```

Out[60]:

	id	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	c
0	0	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	0.916659	0
1	1	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	0.699993	0
2	2	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0	0.399992	0.333328	0.399992	0.249997	0.399996	0
3	3	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0	0.000000	0.000000	0.000000	0.000000	0.000000	0
4	4	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.666644	0.571420	0

5 rows × 30 columns

In [61]:

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

Out[61]:

```
(404290, 28)
```

4. Splitting data into train and test

In [0]:

```
features =['cwc_min','cwc_max','csc_min','csc_max','ctc_min','ctc_max','last_word_eq','firs
x=df[features]
y = y_true
```

In [0]:

```
from sklearn.model_selection import train_test_split

X_train,X_test, y_train, y_test = train_test_split(x, y, stratify=y, test_size=0.3)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_si
```

Knowing Feature Importance

In [64]:

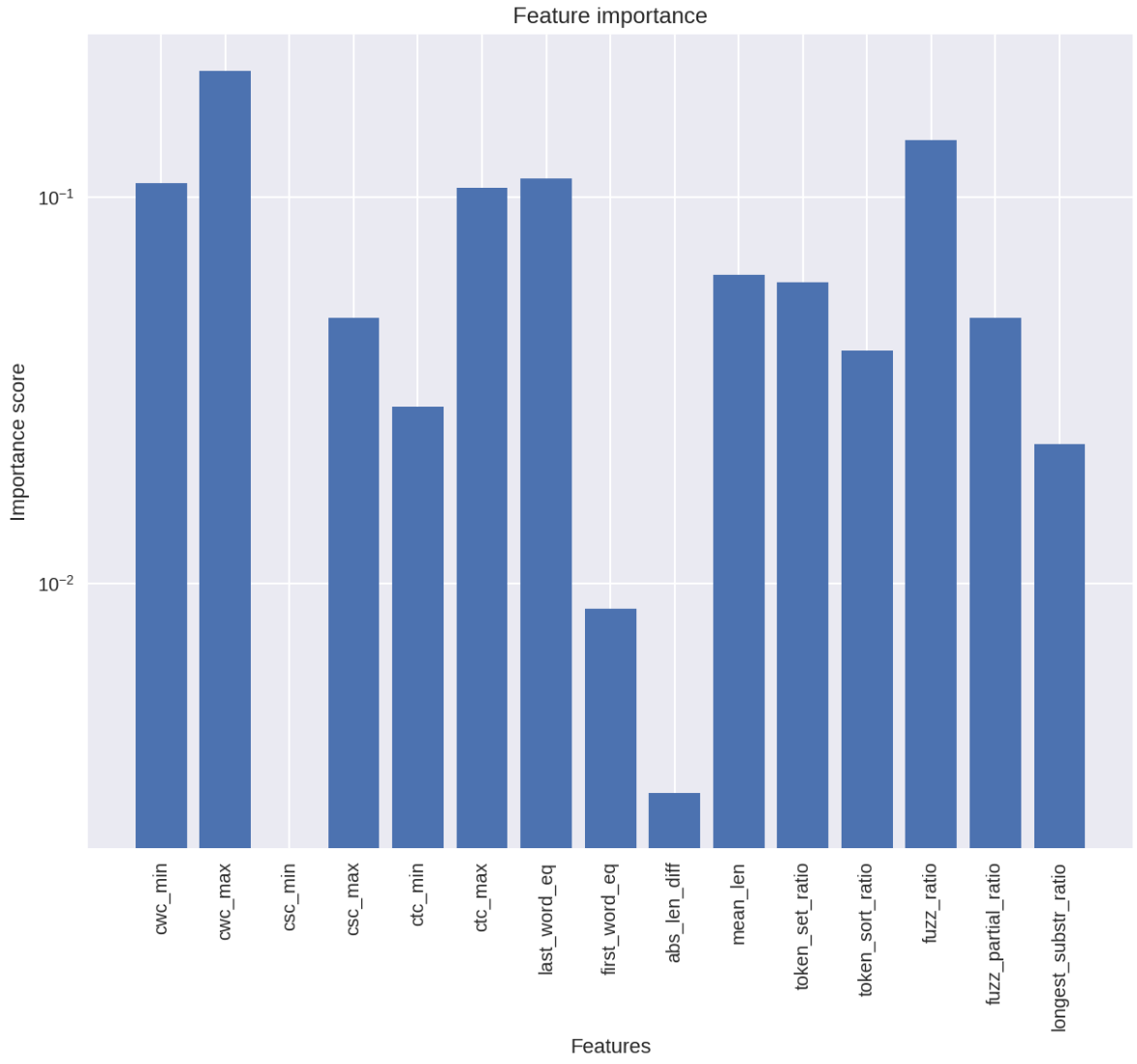
```
from xgboost import XGBClassifier
model = XGBClassifier(n_estimators=50)
model.fit(X_train,y_train)
```

Out[64]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=50,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [65]:

```
feature_importance = model.feature_importances_

plt.figure(figsize=(10,8))
plt.yscale('log', nonposy='clip')

#plt.tight_layout()
plt.bar(range(len(feature_importance)), feature_importance, align='center')
plt.xticks(range(len(feature_importance)),features, rotation='vertical')
plt.title('Feature importance')
plt.ylabel('Importance score')
plt.xlabel('Features')
plt.show()
```



In [66]:

```
feature_importances = pd.DataFrame(model.feature_importances_, columns=['weights'], index=f
feature_importances
```

Out[66]:

	weights
cwc_max	0.211429
fuzz_ratio	0.140000
last_word_eq	0.111429
cwc_min	0.108571
ctc_max	0.105714
mean_len	0.062857
token_set_ratio	0.060000
csc_max	0.048571
fuzz_partial_ratio	0.048571
token_sort_ratio	0.040000
ctc_min	0.028571
longest_substr_ratio	0.022857
first_word_eq	0.008571
abs_len_diff	0.002857
csc_min	0.000000

4.2 Selecting Important Features

In [0]:

```
def selecting_Important_features(feature_importances, min_score):
    column_slice = feature_importances[feature_importances['weights'] > min_score]
    return column_slice.index.values
```

In [68]:

```
Topfeatures = selecting_Important_features(feature_importances, 0.002)
print(' Topfeatures:', Topfeatures)
```

```
 Topfeatures: ['cwc_max' 'fuzz_ratio' 'last_word_eq' 'cwc_min' 'ctc_max' 'me
an_len'
 'token_set_ratio' 'csc_max' 'fuzz_partial_ratio' 'token_sort_ratio'
 'ctc_min' 'longest_substr_ratio' 'first_word_eq' 'abs_len_diff']
```

In [69]:

```
df = df.drop(['abs_len_diff','csc_min'],axis=1)
df.head()
```

Out[69]:

	question1	question2	cwc_min	cwc_max	csc_max	ctc_min	ctc_max	last_word_eq	first
0	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0.999980	0.833319	0.999983	0.916659	0.785709	0.0	
1	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0.799984	0.399996	0.599988	0.699993	0.466664	0.0	
2	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0.399992	0.333328	0.249997	0.399996	0.285712	0.0	
3	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	
4	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0.399992	0.199998	0.666644	0.571420	0.307690	0.0	

5 rows × 26 columns

In [0]:

```
from sklearn.model_selection import train_test_split

X_train,X_test, y_train, y_test = train_test_split(df, y_true, stratify=y_true, test_size=0
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_si
```

In [71]:

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

```
Number of data points in train data : (226402, 26)
Number of data points in cross validation data: (56601, 26)
Number of data points in test data : (121287, 26)
```

4.3 Tf-idf Vectorization

In [0]:

```
tfidf = TfidfVectorizer()
train_questions = X_train['question1'] + X_train['question2']
tfidf_train_feature = tfidf.fit_transform(train_questions.values.astype('U'))
X_train = normalize(tfidf_train_feature,axis = 0)
cv_questions = X_cv['question1'] + X_cv['question2']
tfidf_cv_feature = tfidf.transform(cv_questions.values.astype('U'))
X_cv = normalize(tfidf_cv_feature,axis = 0)
te_questions = X_test['question1'] + X_test['question2']
tfidf_test_feature = tfidf.transform(te_questions.values.astype('U'))
X_test = normalize(tfidf_test_feature,axis = 0)
```

4.4 Distribution yi's

In [73]:

```
from collections import Counter

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

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308027314246341 Class 1:  0.36919726857536594
---------- Distribution of output variable in train data ----------
Class 0:  0.630801575943888 Class 1:  0.3691984240561121
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

4.5 Confusion Matrix

In [0]:

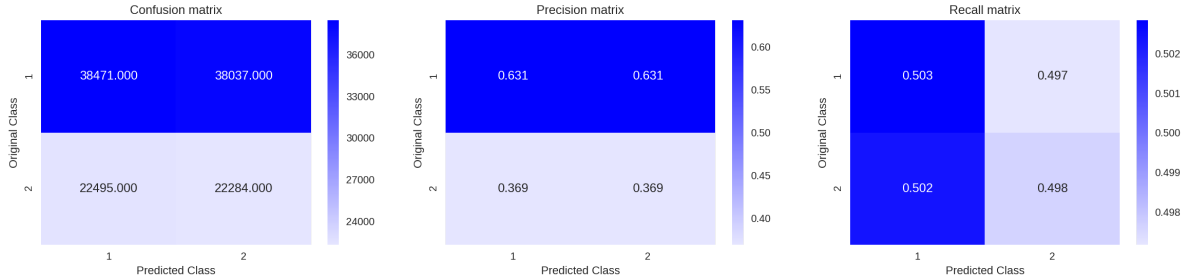
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(y_test, predict_y):
    C = confusion_matrix(y_test, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted

    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 a
    # 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 a
    # 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",as_cmap=True)
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

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

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

    plt.show()
```

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

In [75]:

```
# 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 0.8856842905615624
```



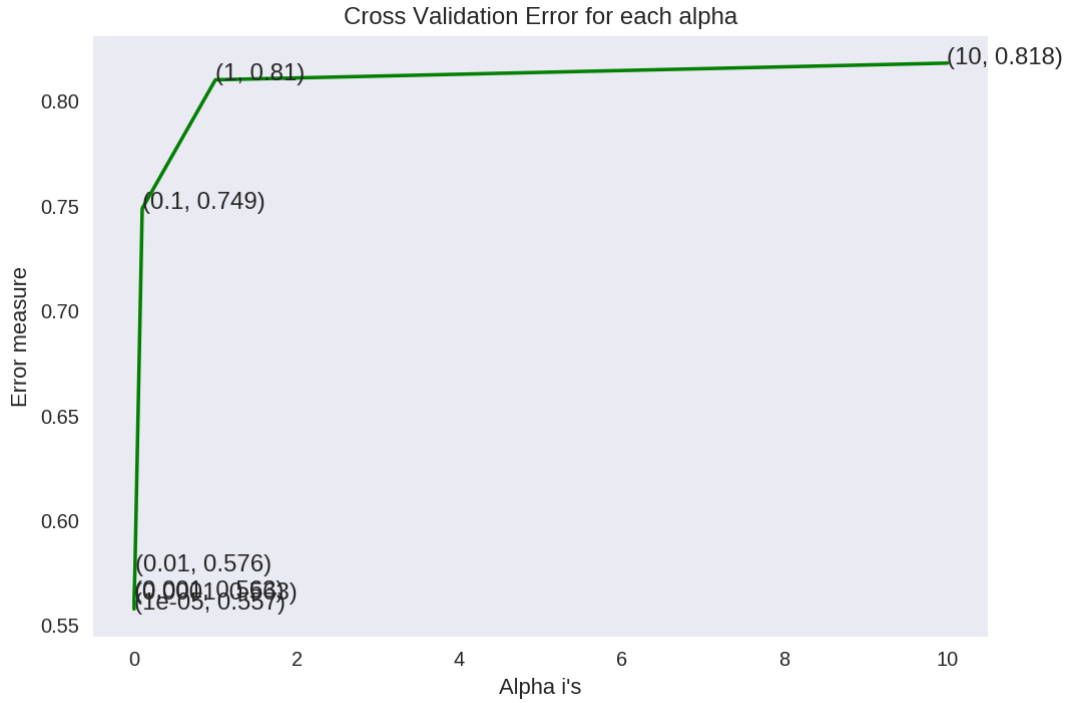
4.7 Logistic Regression with hyperparameter tuning

In [76]:

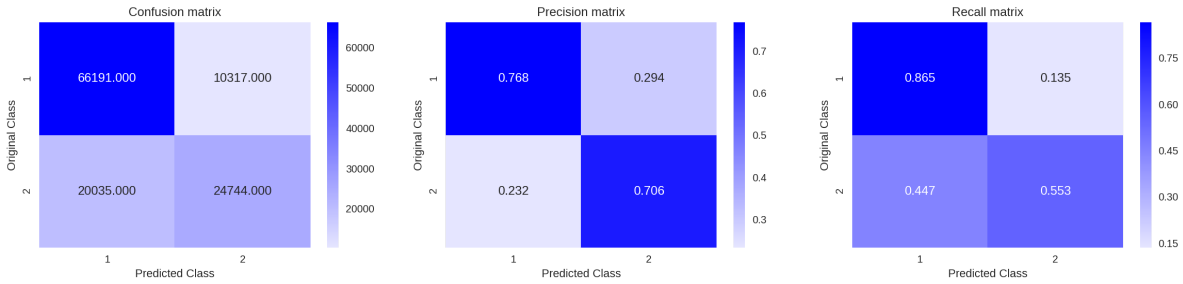
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cv log loss is:",log_loss(y_cv
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)
```

```
For values of alpha =  1e-05 The log loss is: 0.5574345113923482
For values of alpha =  0.0001 The log loss is: 0.5625148445194551
For values of alpha =  0.001 The log loss is: 0.5631994095338251
```

```
For values of alpha =  0.01 The log loss is: 0.5756310637771552
For values of alpha =  0.1 The log loss is: 0.7485887856479101
For values of alpha =  1 The log loss is: 0.8099533729429392
For values of alpha =  10 The log loss is: 0.8179111407354543
```




```
For values of best alpha =  1e-05 The train log loss is: 0.44147864128312814
For values of best alpha =  1e-05 The cv log loss is: 0.5574345113923482
For values of best alpha =  1e-05 The test log loss is: 0.5146439453988134
Total number of data points : 121287
```



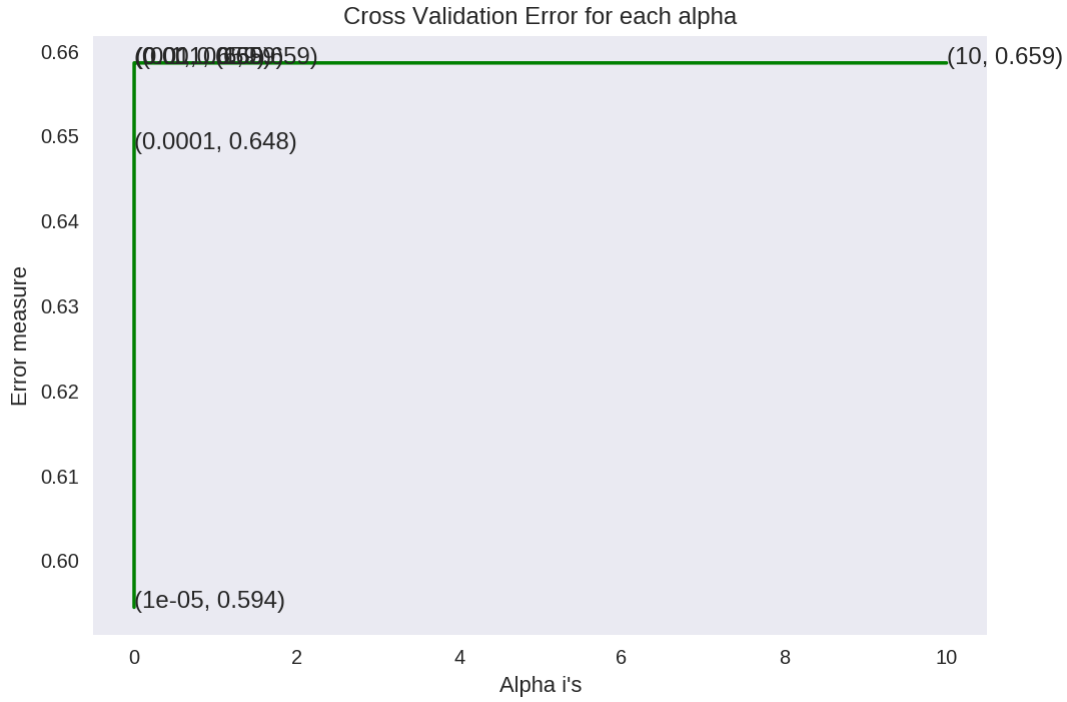
4.8 Linear SVM with hyperparameter tuning

In [77]:

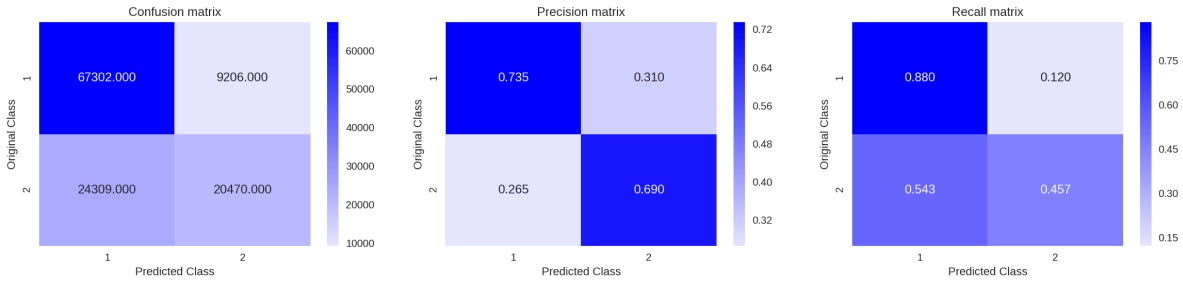
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cv log loss is:",log_loss(y_cv
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)
```

```
For values of alpha =  1e-05 The log loss is: 0.5944102801048881
For values of alpha =  0.0001 The log loss is: 0.6484697496292379
For values of alpha =  0.001 The log loss is: 0.658527689864188
```

```
For values of alpha =  0.01 The log loss is: 0.658527689864188
For values of alpha =  0.1 The log loss is: 0.658527689864188
For values of alpha =  1 The log loss is: 0.658527689864188
For values of alpha =  10 The log loss is: 0.658527689864188
```




```
For values of best alpha =  1e-05 The train log loss is: 0.5441989263918059
For values of best alpha =  1e-05 The cv log loss is: 0.5944102801048881
For values of best alpha =  1e-05 The test log loss is: 0.5638897363109426
Total number of data points : 121287
```



4.9 XGBoost

In [78]:

```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,cv=2,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  5.9min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 19.0min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 25.2min
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 41.5min remaining:
0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 41.5min finished
```

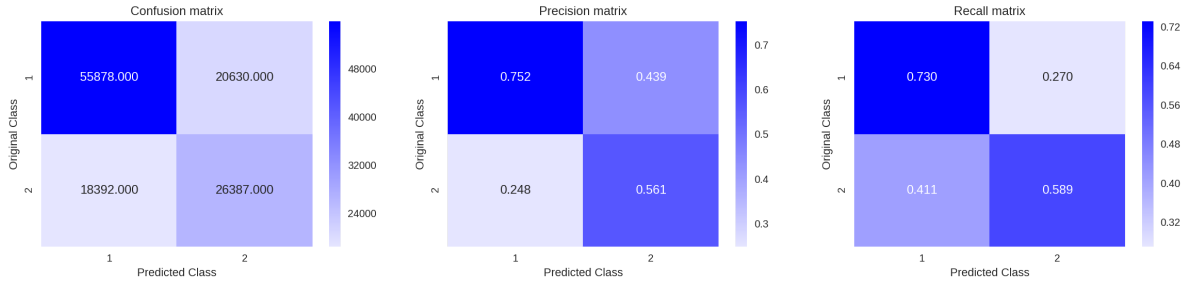
Out[78]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsampl
e_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.1
5, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 1
0], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5,
1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [79]:

```
x_cfl=XGBClassifier(base_score=0.5, booster='gbtree',
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=-1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, 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(
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
plot_confusion_matrix(y_test,sig_clf.predict(X_test))
```

```
For values of best alpha =  1e-05 The train log loss is: 0.5465301559205789
For values of best alpha =  1e-05 The cross validation log loss is: 0.694409
941091639
For values of best alpha =  1e-05 The test log loss is: 0.5914038659351692
```



5 Model Performance Table

In [82]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.add_column("Sno.",[1,2,3,4])
x.add_column("Model",['Random','Logistic_Regression','Linear SVM','Xgboost'])
x.add_column("Hyperparameter",['-','1e-05','1e-05','1e-05'])
x.add_column("Train Logoss",['-','0.4414','0.5441','0.5465'])
x.add_column("CV Logoss",['-','0.55574','0.5944','06944'])
x.add_column("Test Log Loss",['0.8858','0.5146','0.5638','0.5914'])
print(x)
```

```
+------+---------------------+---------------+-------------+-----------+--
-------------+
| Sno. |        Model        | Hyperparameter | Train Logoss | CV Logoss | T
est Log Loss |
+------+---------------------+---------------+-------------+-----------+--
-------------+
|  1   |        Random       |       -       |      -       |     -     |
0.8858    |
|  2   | Logistic_Regression |     1e-05     |    0.4414    |  0.55574  |
0.5146    |
|  3   |      Linear SVM     |     1e-05     |    0.5441    |   0.5944  |
0.5638    |
|  4   |       Xgboost       |     1e-05     |    0.5465    |   06944   |
0.5914    |
+------+---------------------+---------------+-------------+-----------+--
-------------+
```

<h2> 6.Conclusion

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis

algorithm and based on the concept of probability.Logistic regression is a very powerful algorithm, even for very complex

problems it may do a good job.We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a

more complex cost function, this cost function can be defined as the ‘Sigmoid function’ or also known as the ‘logistic function’

instead of a linear function.The hypothesis of logistic regression tends it to limit the cost function between 0 and 1.

Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as

per the hypothesis of logistic regression.In order to map predicted values to probabilities, we use the Sigmoid function. The

function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to

probabilities.

When using linear regression we used a formula of the hypothesis i.e.

$h\Theta(x) = \beta_0 + \beta_1 X$

For logistic regression we are going to modify it a little bit i.e.

$\sigma(Z) = \sigma(\beta_0 + \beta_1 X)$

We have expected that our hypothesis will give values between 0 and 1.

gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model.

we can estimate the importance of features for a predictive modeling problem using the XGBoost.

How feature importance is calculated using the gradient boosting algorithm.

How to plot feature importance in Python calculated by the XGBoost model.

How to use feature importance calculated by XGBoost to perform feature selection. all these ideas are implemented here.

Steps Involved:-

1) understanding bussiness problem objectives and constraints and load Necessary files

2) we come to know whether it classification problem or Regression problem

3) Exploratory Data Analysis(basic feature extraction,we plot univariate analysis,pairplot,checking duplicates,checking null values,filling null values,Analysis of some extracted features,boxplot,joint plots, checking outliers)

4) calculating mean,median,percentiles etc

5) preprocessiong of text(it includes removing html tags,removing punctutations,performing stemming,removinf stopwords,contraction etc)

6) Checking correlation

7) Splitting data into train (80%) and Test (20%)

8) Building feature importance by xgboost (we can select those feature for building models)

9) Tf-idf Vectorization

10) Building a Random Model For comparision (when i built new model that model should be better than this)

11) Building logistic Regression ,Linear Svm and Xgboost models with Hyperparameter Tunning

12) Checking model is overfitting or not

13)Comparing Results by model performance table

by checking confusion matrix and loglosses logistic regression model is showing better results

- so here My best model here is logistic Regression it has low test loglosss

In [ ]:

```
```