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 (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 [0]:

```
!pip install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437ae
fa0d0a92ccfb38895d9416bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz (https://
files.pythonhosted.org/packages/5c/1a/883e47df323437aefa0d0a92ccfb38895d9416
bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz) (180kB)
    100% |████████████████████████████████| 184kB 6.8MB/s
Building wheels for collected packages: distance
  Building wheel for distance (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d0f
12db1c66dbae9c5442b39b001db18e
Successfully built distance
Installing collected packages: distance
Successfully installed distance-0.1.3
```

In [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 plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

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

In [4]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2
F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.co
m%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.re
adonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&respon
se_type=code (https://accounts.google.com/o/oauth2/auth?client_id=9473189898
03-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=
urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleap
is.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3
A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
..........
Mounted at /content/gdrive
```

In [5]:

```
# setting path
import os
# par_path = os.path.normpath(os.getcwd() + os.sep + os.pardir)
dir_path = '/content/gdrive/My Drive/appliedAI/quora_new/'
dir_path
```

Out[5]:

```
'/content/gdrive/My Drive/appliedAI/quora_new/'
```

3.1 Reading data and basic stats

In [0]:

```
df = pd.read_csv(os.path.join(dir_path, "train.csv"))

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

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

In [0]:

```
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 [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
```

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

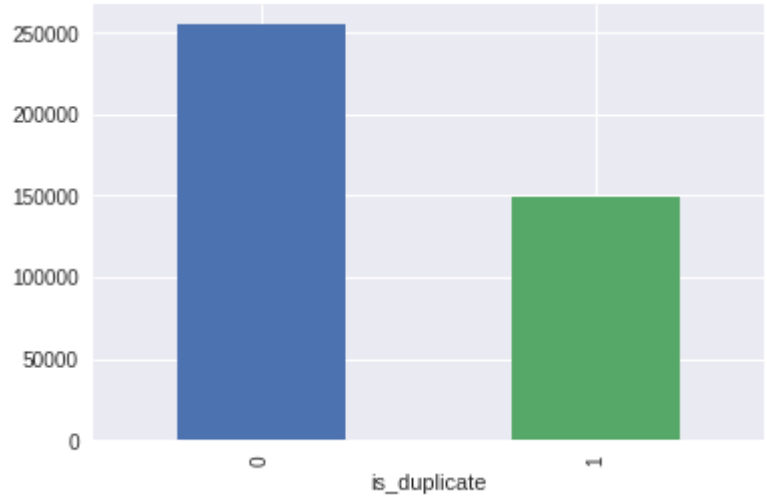
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [0]:

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

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46c014b860>
```



In [0]:

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

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

In [0]:

```
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

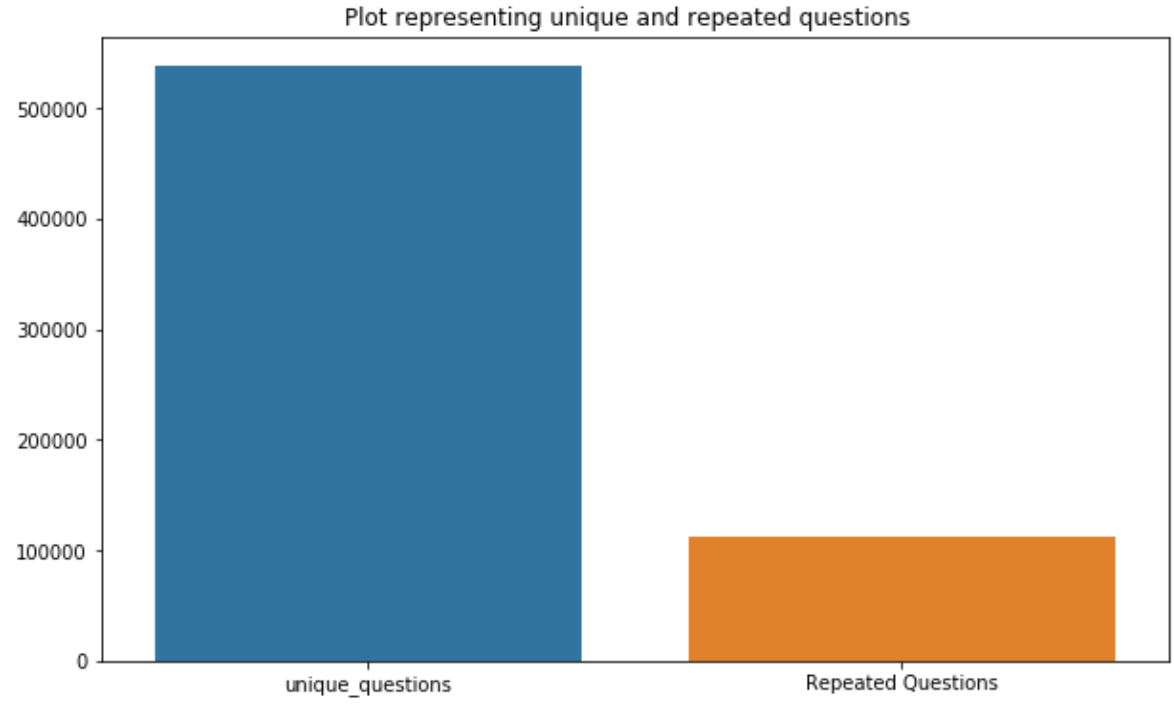
In [0]:

```
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_cour

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 (20.779539
45937505%)

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

In [0]:

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

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



3.2.3 Checking for Duplicates

In [0]:

```
#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])
```

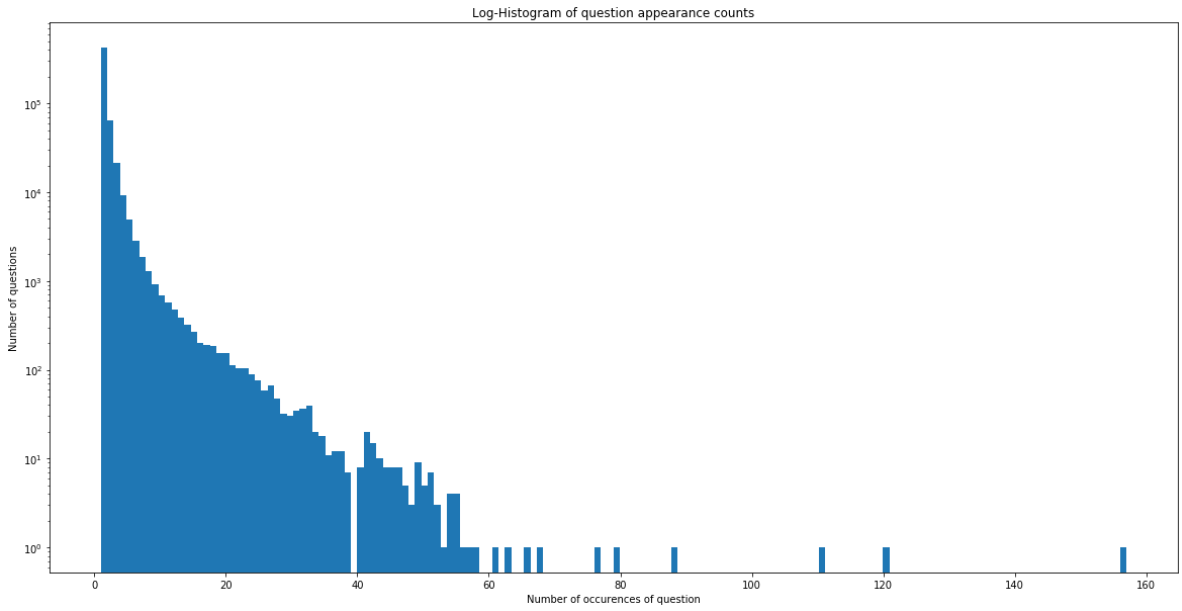
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [0]:

```
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

In [0]:

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

```
            id    qid1    qid2                         question1 question2
\
105780  105780  174363  174364    How can I develop android app?       NaN
201841  201841  303951  174364  How can I create an Android app?       NaN

        is_duplicate
105780             0
201841             0
```

- There are two rows with null values in question2

In [0]:

```
# 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 [0]:

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

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

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

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

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

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

df.head()
```

Out[20]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2le
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

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2le
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 [0]:

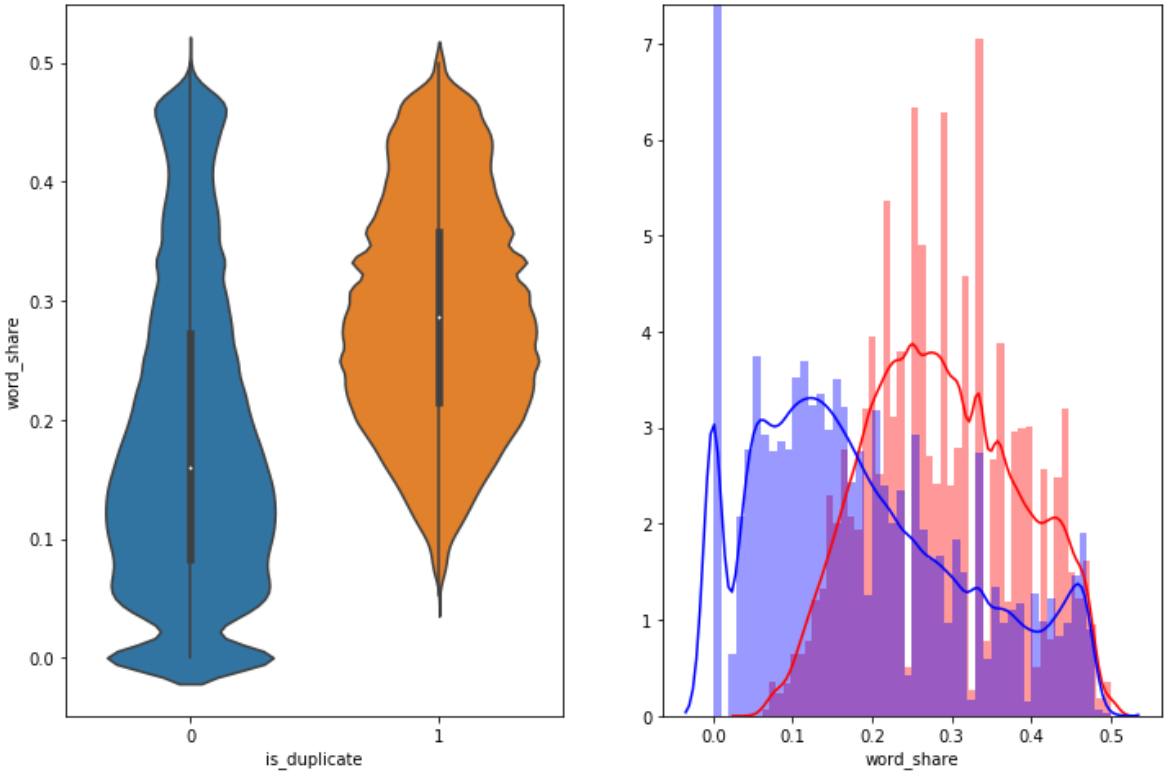
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

In [0]:

```
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()
```

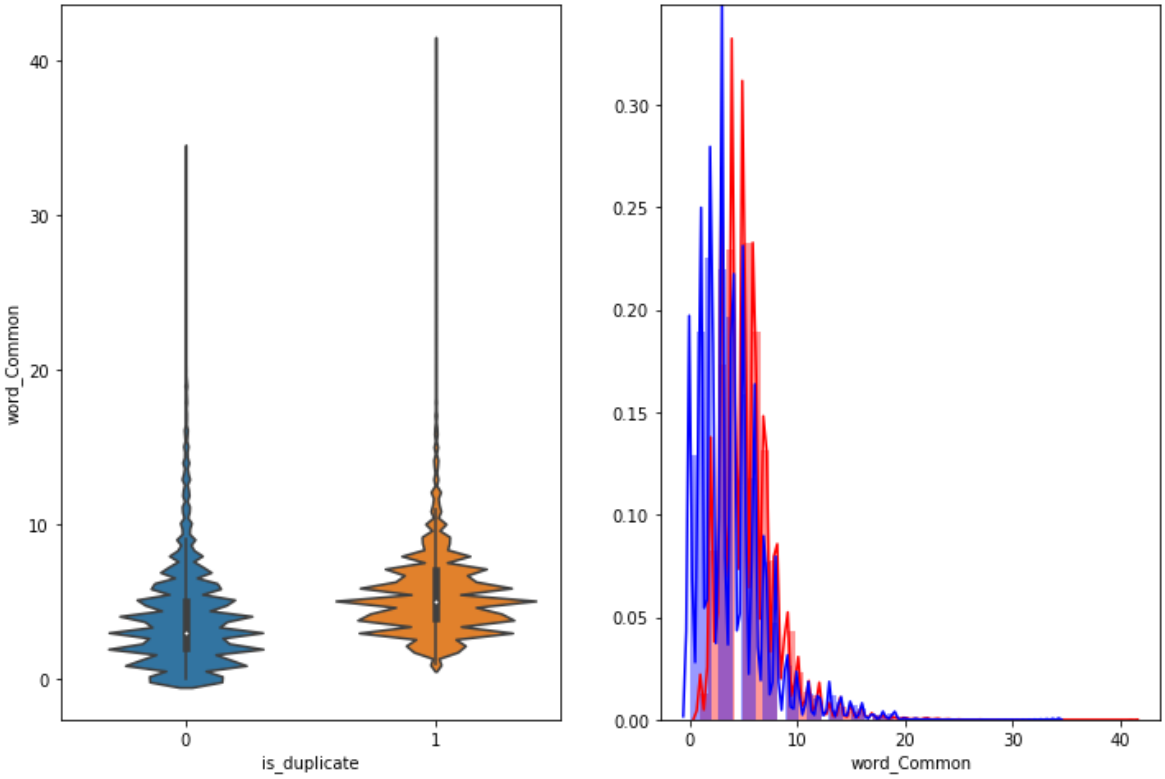


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [0]:

```
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()
```



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

1.2.1 : EDA: Advanced Feature Extraction.

In [0]:

```
!pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7ea
a1beab2b9323073815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-none-
any.whl (https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7eaa1bea
b2b9323073815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-none-any.w
hl)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

In [0]:

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

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

In [0]:

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byt
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous noteboo
```

In [0]:

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

Out[8]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len
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 [0]:

```
# 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
```

- 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_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2
  cwc_min = common_word_count / (min(len(q1_words), len(q2_words))

- **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
  cwc_max = common_word_count / (max(len(q1_words), len(q2_words))

- **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2
  csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))

- **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))

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

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

- **last_word_eq** : Check if Last 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]:

```
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 [0]:

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

Out[12]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc
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	0.9
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	0.5

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 [0]:

```
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
```

In [0]:

```
# reading the text files and removing the Stop Words:
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))
```

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193130
```

Word Clouds generated from duplicate pair question's text

In [0]:

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

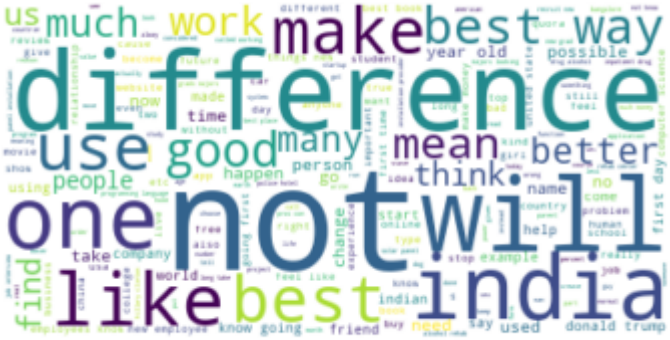
```
Word Cloud for Duplicate Question pairs
```

Word Clouds generated from non duplicate pair question's text

In [0]:

```
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()
```

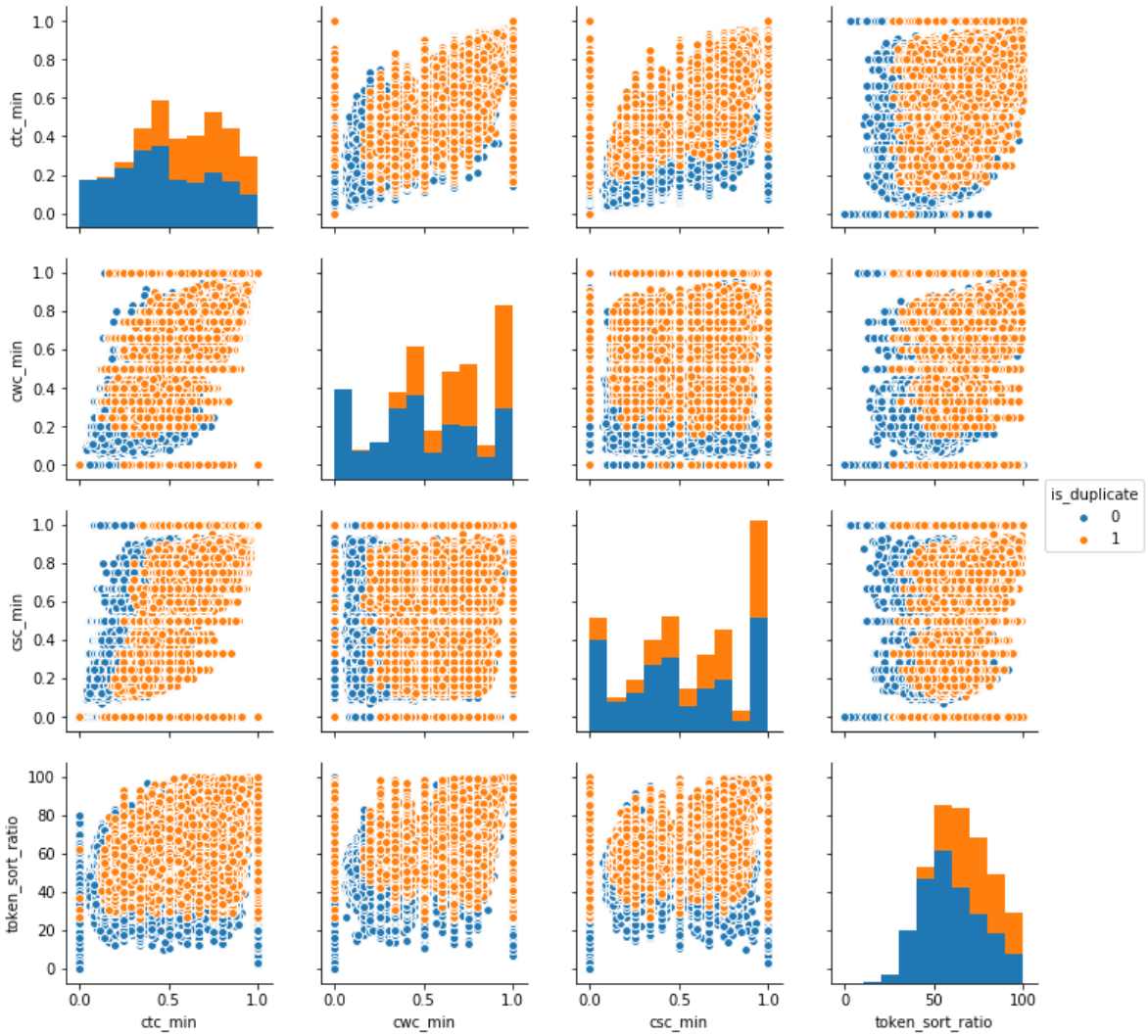
Word Cloud for non-Duplicate Question pairs:



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

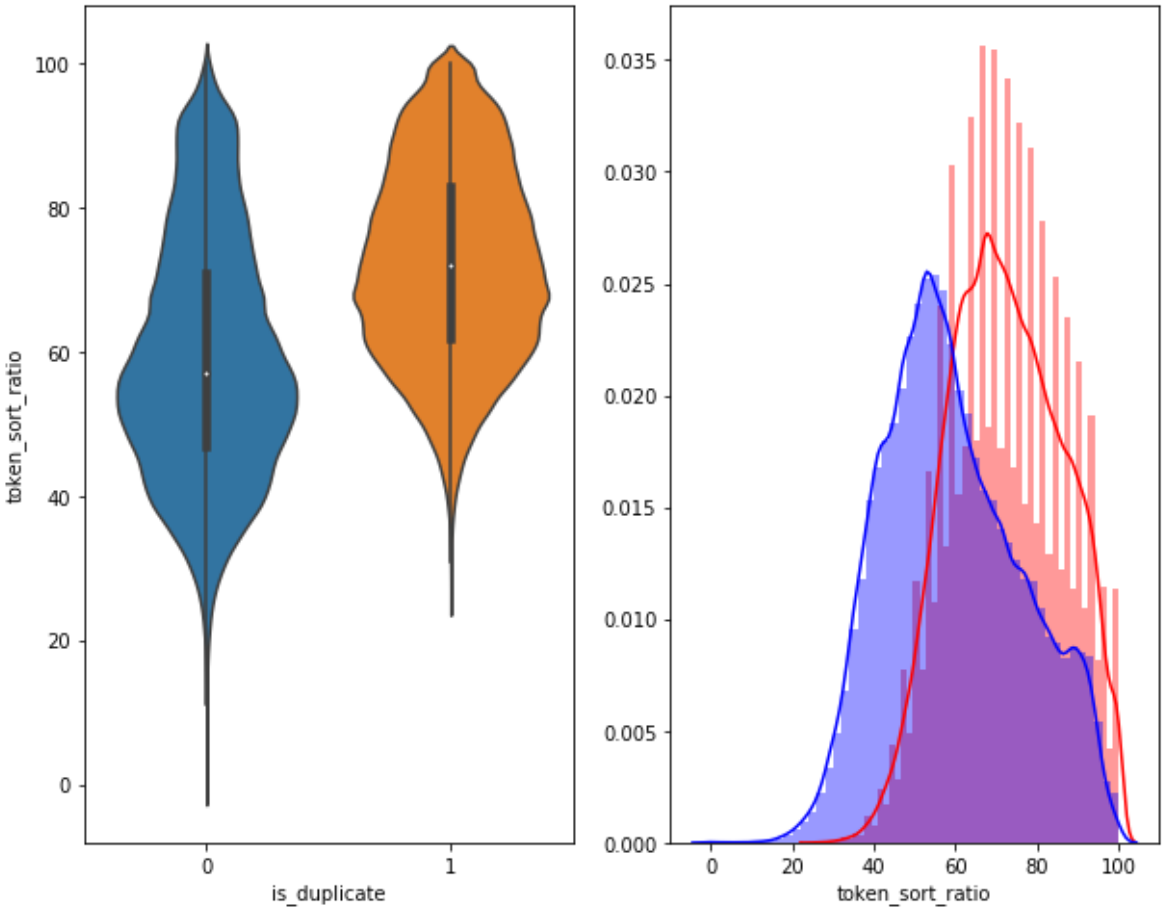
In [0]:

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



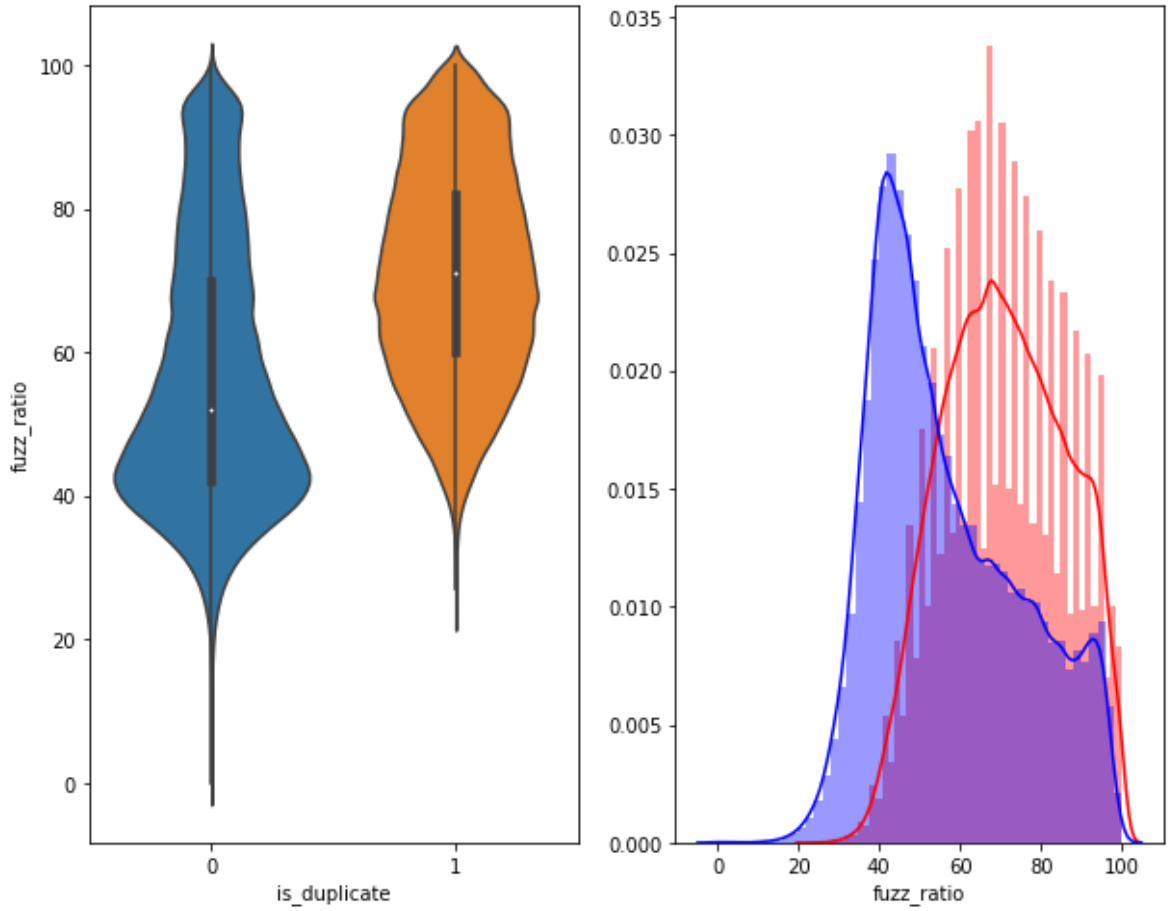
In [0]:

```
# 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()
```



In [0]:

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

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

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



3.5.2 Visualization

In [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 [0]:

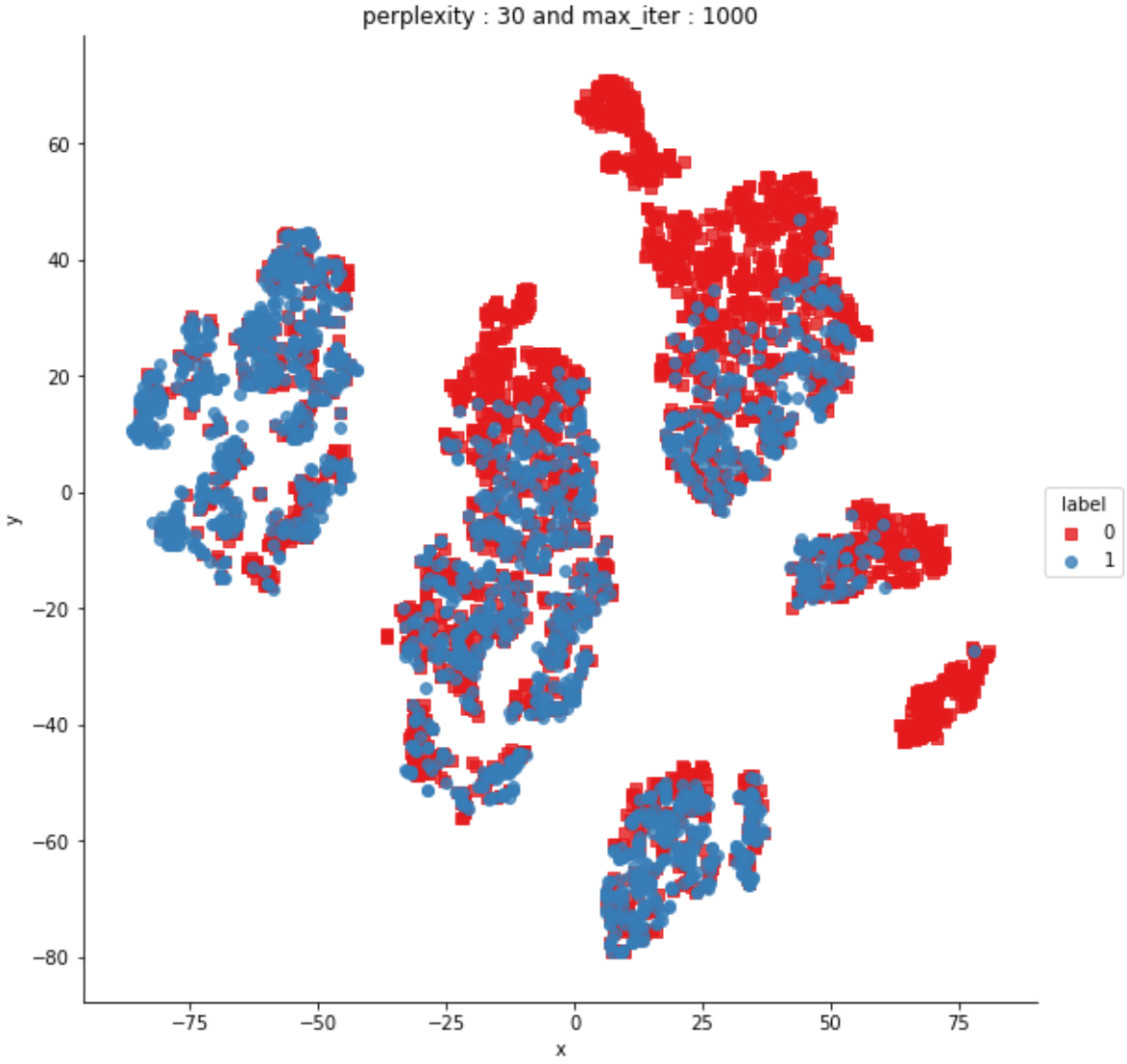
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.011s...
[t-SNE] Computed neighbors for 5000 samples in 0.912s...
[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 0.433s
[t-SNE] Iteration 50: error = 80.9244080, gradient norm = 0.0428133 (50 iter
ations in 13.099s)
[t-SNE] Iteration 100: error = 70.3858795, gradient norm = 0.0100968 (50 ite
rations in 9.067s)
[t-SNE] Iteration 150: error = 68.6138382, gradient norm = 0.0058392 (50 ite
rations in 9.602s)
[t-SNE] Iteration 200: error = 67.7700119, gradient norm = 0.0036596 (50 ite
rations in 9.121s)
[t-SNE] Iteration 250: error = 67.2725067, gradient norm = 0.0034962 (50 ite
rations in 11.305s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.27250
7
[t-SNE] Iteration 300: error = 1.7737305, gradient norm = 0.0011918 (50 iter
ations in 8.289s)
[t-SNE] Iteration 350: error = 1.3720417, gradient norm = 0.0004822 (50 iter
ations in 10.526s)
[t-SNE] Iteration 400: error = 1.2039998, gradient norm = 0.0002768 (50 iter
ations in 9.600s)
[t-SNE] Iteration 450: error = 1.1133438, gradient norm = 0.0001881 (50 iter
ations in 11.827s)
[t-SNE] Iteration 500: error = 1.0579143, gradient norm = 0.0001434 (50 iter
ations in 8.941s)
[t-SNE] Iteration 550: error = 1.0221983, gradient norm = 0.0001164 (50 iter
ations in 11.092s)
[t-SNE] Iteration 600: error = 0.9987167, gradient norm = 0.0001039 (50 iter
ations in 11.467s)
[t-SNE] Iteration 650: error = 0.9831534, gradient norm = 0.0000938 (50 iter
ations in 11.799s)
[t-SNE] Iteration 700: error = 0.9722011, gradient norm = 0.0000858 (50 iter
ations in 12.028s)
[t-SNE] Iteration 750: error = 0.9643636, gradient norm = 0.0000799 (50 iter
ations in 12.120s)
[t-SNE] Iteration 800: error = 0.9584482, gradient norm = 0.0000785 (50 iter
ations in 11.867s)
[t-SNE] Iteration 850: error = 0.9538348, gradient norm = 0.0000739 (50 iter
ations in 11.461s)
[t-SNE] Iteration 900: error = 0.9496906, gradient norm = 0.0000712 (50 iter
ations in 11.023s)
[t-SNE] Iteration 950: error = 0.9463405, gradient norm = 0.0000673 (50 iter
```

```
ations in 11.755s)
[t-SNE] Iteration 1000: error = 0.9432716, gradient norm = 0.0000662 (50 ite
rations in 11.493s)
[t-SNE] Error after 1000 iterations: 0.943272
```

In [0]:

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

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



3.6 Featurizing text data with tfidf weighted word-vectors

In [0]:

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

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

In [0]:

```
# avoid decoding problems
df = pd.read_csv(os.path.join(dir_path, "train.csv"))

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

In [0]:

```
df.head()
```

Out[7]:

	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 [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

In [0]:

```
# merge texts
quest_1_80k = df['question1'].head(80000).copy()
quest_1_20k = df['question1'][80000:100000].copy()
quest_2_80k = df['question2'].head(80000).copy()
quest_2_20k = df['question2'][80000:100000].copy()

questions_train = list(quest_1_80k) + list(quest_2_80k)

tfidf = TfidfVectorizer(ngram_range=(1,2))
tfidf.fit(questions_train)
```

```
(80000, 438024) (20000, 438024) (80000, 438024) (20000, 438024)
```

In [0]:

```
q1_80k = tfidf.transform(quest_1_80k.values.tolist())
q1_20k = tfidf.transform(quest_1_20k.values.tolist())
q2_80k = tfidf.transform(quest_2_80k.values.tolist())
q2_20k = tfidf.transform(quest_2_20k.values.tolist())
# dict key:word and value:tf-idf score
# word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
print(q1_80k.shape, q1_20k.shape, q2_80k.shape, q2_20k.shape)
```

```
(80000, 438024) (20000, 438024) (80000, 438024) (20000, 438024)
```

In [0]:

```
df = df.head(100000)
```

In [0]:

```
from scipy.sparse import hstack
```

In [0]:

```
x_train_quests = hstack((q1_80k, q2_80k))
x_test_quests = hstack((q1_20k, q2_20k))
print(x_train_quests.shape, x_test_quests.shape)
```

```
(80000, 876048) (20000, 876048)
```

In [0]:

```
df.columns
```

Out[24]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate'], dtyp
e='object')
```

In [0]:

```
y_train = df['is_duplicate'][:80000].values.tolist()
y_test = df['is_duplicate'][80000:100000].values.tolist()
print(len(y_train), len(y_test))
```

```
80000 20000
```

In [0]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile(os.path.join(dir_path, 'nlp_features_train.csv')):
    dfnlp = pd.read_csv(os.path.join(dir_path, "nlp_features_train.csv"),encoding='latin-1'
else:
    print("download nlp_features_train.csv from drive or run previous notebook")

if os.path.isfile(os.path.join(dir_path, 'df_fe_without_preprocessing_train.csv')):
    dfppro = pd.read_csv(os.path.join(dir_path, "df_fe_without_preprocessing_train.csv"),er
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebo
```

In [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)
# df3_q1 = pd.DataFrame(df3['q1_feats_m'].values.tolist(), index= df3.index)
# df3_q2 = pd.DataFrame(df3['q2_feats_m'].values.tolist(), index= df3.index)
print(df1.columns, df2.columns, df3.columns)
```

```
Index(['id', 'is_duplicate', 'cwc_min', 'cwc_max', 'csc_min', 'csc_max',
       'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq', 'abs_len_dif
f',
       'mean_len', 'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object') Index(['id', 'freq_qid1', 'freq_qid2', 'q1len', 'q2le
n', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share', 'freq_q1+q
2',
       'freq_q1-q2'],
      dtype='object') Index(['id'], dtype='object')
```

In [0]:

```
df1 = df1.drop(['id', 'is_duplicate'], axis=1)
df2 = df2.drop(['id'], axis=1)
print(df1.columns)
print(df2.columns)
print(df3.columns)
```

```
Index(['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')
Index(['freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_word
s',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q
2'],
      dtype='object')
Index(['id'], dtype='object')
```

In [0]:

```
# dataframe of nlp features
df1.head()
```

Out[32]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0

In [0]:

```
# data before preprocessing
df2.head()
```

Out[33]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_T
0	1	1	66	57	14	12	10.0	23.0
1	4	1	51	88	8	13	4.0	20.0
2	1	1	73	59	14	10	4.0	24.0
3	1	1	50	65	11	9	0.0	19.0
4	3	1	76	39	13	7	2.0	20.0

In [0]:

```
print(df1.shape, df2.shape)
```

```
(404290, 15) (404290, 11)
```

In [0]:

```
from scipy import sparse
```

In [0]:

```
feat_1_train = sparse.csr_matrix(df1[:80000].to_sparse())
feat_1_test = sparse.csr_matrix(df1[80000:100000].to_sparse())
feat_2_train = sparse.csr_matrix(df2[:80000].to_sparse())
feat_2_test = sparse.csr_matrix(df2[80000:100000].to_sparse())
print(feat_1_train.shape, feat_1_test.shape, feat_2_train.shape, feat_2_test.shape)
```

```
(80000, 15) (20000, 15) (80000, 11) (20000, 11)
```

In [0]:

```
final_train = hstack((feat_1_train, feat_2_train, x_train_quests))
final_test = hstack((feat_1_test, feat_2_test, x_test_quests))
print(final_train.shape, final_test.shape)
```

```
(80000, 876074) (20000, 876074)
```

In [0]:

```
print("Number of features in nlp data:", feat_1_train.shape[1])
print("Number of features in preprocessed data :", feat_2_train.shape[1])
print("Number of features in question1 data :", q1_80k.shape[1])
print("Number of features in question2 data :", q2_80k.shape[1])
print("Number of features in final data  :", final_train.shape[1])
```

```
Number of features in nlp data: 15
Number of features in preprocessed data : 11
Number of features in question1 data : 438024
Number of features in question2 data : 438024
Number of features in final data  : 876074
```

In [0]:

```
# storing the final features to pkl file
import pickle
if not os.path.isfile(os.path.join(dir_path, 'final_features.pkl')):
    with open(os.path.join(dir_path, 'final_features.pkl'), 'wb') as f:
        final_data = {}
        final_data['final_train'] = final_train
        final_data['final_test'] = final_test
        final_data['y_train'] = y_train
        final_data['y_test'] = y_test
        pickle.dump(final_data, f)
```

In [6]:

```
# restoring final features from pkl file
import pickle
if os.path.isfile(os.path.join(dir_path, 'final_features.pkl')):
    with open(os.path.join(dir_path, 'final_features.pkl'), 'rb') as f:
        final_data = pickle.load(f)
final_data.keys()
```

Out[6]:

```
dict_keys(['final_train', 'final_test', 'y_train', 'y_test'])
```

[4] ML Models

In [0]:

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

```
from xgboost import XGBClassifier
```

In [0]:

```
# 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 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")
    cmap=sns.light_palette("Navy", 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()
```

[4.1] Hyperparameter tuning XGBoost using

RandomizedSearch

In [0]:

```
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
n_estimators = [5, 10, 50, 100, 200, 500, 1000]
param_grid = {"max_depth": max_depth, "n_estimators": n_estimators}
```

In [0]:

```
grid = RandomizedSearchCV(
    XGBClassifier(n_jobs=-1), param_grid,
    scoring='neg_log_loss', verbose=2, cv=2
)
grid.fit(final_train, y_train)

print("Best Params:", grid.best_params_)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV] n_estimators=500, max_depth=3 ....................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.

[CV] ..................... n_estimators=500, max_depth=3, total= 8.4min
[CV] n_estimators=500, max_depth=3 ....................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  8.4min remaining:
0.0s

[CV] ..................... n_estimators=500, max_depth=3, total= 8.3min
[CV] n_estimators=5, max_depth=4 ......................................
[CV] ....................... n_estimators=5, max_depth=4, total=  10.1s
[CV] n_estimators=5, max_depth=4 ......................................
[CV] ....................... n_estimators=5, max_depth=4, total=   9.6s
[CV] n_estimators=1000, max_depth=5 ...................................
[CV] ................... n_estimators=1000, max_depth=5, total=27.8min
[CV] n_estimators=1000, max_depth=5 ...................................
[CV] ................... n_estimators=1000, max_depth=5, total=27.6min
[CV] n_estimators=1000, max_depth=2 ...................................
[CV] ................... n_estimators=1000, max_depth=2, total=11.0min
[CV] n_estimators=1000, max_depth=2 ...................................
[CV] ................... n_estimators=1000, max_depth=2, total=11.0min
[CV] n_estimators=50, max_depth=4 .....................................
[CV] ...................... n_estimators=50, max_depth=4, total= 1.2min
[CV] n_estimators=50, max_depth=4 .....................................
[CV] ...................... n_estimators=50, max_depth=4, total= 1.2min
[CV] n_estimators=50, max_depth=5 .....................................
[CV] ...................... n_estimators=50, max_depth=5, total= 1.5min
[CV] n_estimators=50, max_depth=5 .....................................
[CV] ...................... n_estimators=50, max_depth=5, total= 1.5min
[CV] n_estimators=10, max_depth=5 .....................................
[CV] ...................... n_estimators=10, max_depth=5, total=  20.7s
[CV] n_estimators=10, max_depth=5 .....................................
[CV] ...................... n_estimators=10, max_depth=5, total=  20.7s
[CV] n_estimators=10, max_depth=3 .....................................
[CV] ...................... n_estimators=10, max_depth=3, total=  12.8s
[CV] n_estimators=10, max_depth=3 .....................................
[CV] ...................... n_estimators=10, max_depth=3, total=  12.8s
[CV] n_estimators=200, max_depth=4 ....................................
[CV] ..................... n_estimators=200, max_depth=4, total= 4.5min
[CV] n_estimators=200, max_depth=4 ....................................
[CV] ..................... n_estimators=200, max_depth=4, total= 4.5min
[CV] n_estimators=1000, max_depth=4 ...................................
[CV] ................... n_estimators=1000, max_depth=4, total=22.1min
[CV] n_estimators=1000, max_depth=4 ...................................
[CV] ................... n_estimators=1000, max_depth=4, total=22.0min

[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed: 154.6min finished

Best Params: {'n_estimators': 1000, 'max_depth': 5}
```

In [10]:

```
classifier = XGBClassifier(n_jobs=-1, n_estimators=1000, max_depth=5, verbose=2)
classifier.fit(final_data['final_train'], final_data['y_train'])
```

Out[10]:

```
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=5, min_child_weight=1, missing=None, n_estimators=1000,
       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, verbose=2)
```

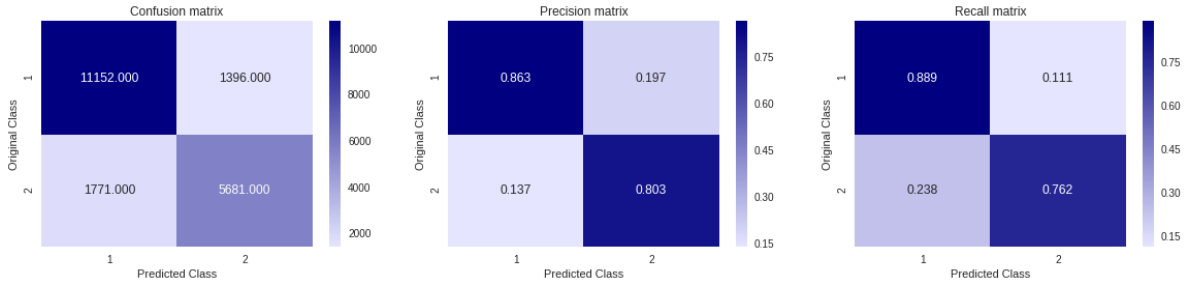
In [11]:

```
predict_y = classifier.predict_proba(final_data['final_train'])
print("The train log loss is:",log_loss(final_data['y_train'], predict_y, labels=classifier

predict_y = classifier.predict_proba(final_data['final_test'])
print("The test log loss is:",log_loss(final_data['y_test'], predict_y, labels=classifier.c
```

```
The train log loss is: 0.25065955738485207
The test log loss is: 0.3233131841806878
```

In [12]:

```
predict_y = np.argmax(predict_y, axis=1)
plot_confusion_matrix(final_data['y_test'], predict_y)
```



[4.2] Hyperparameter tuning Linear SVM

In [0]:

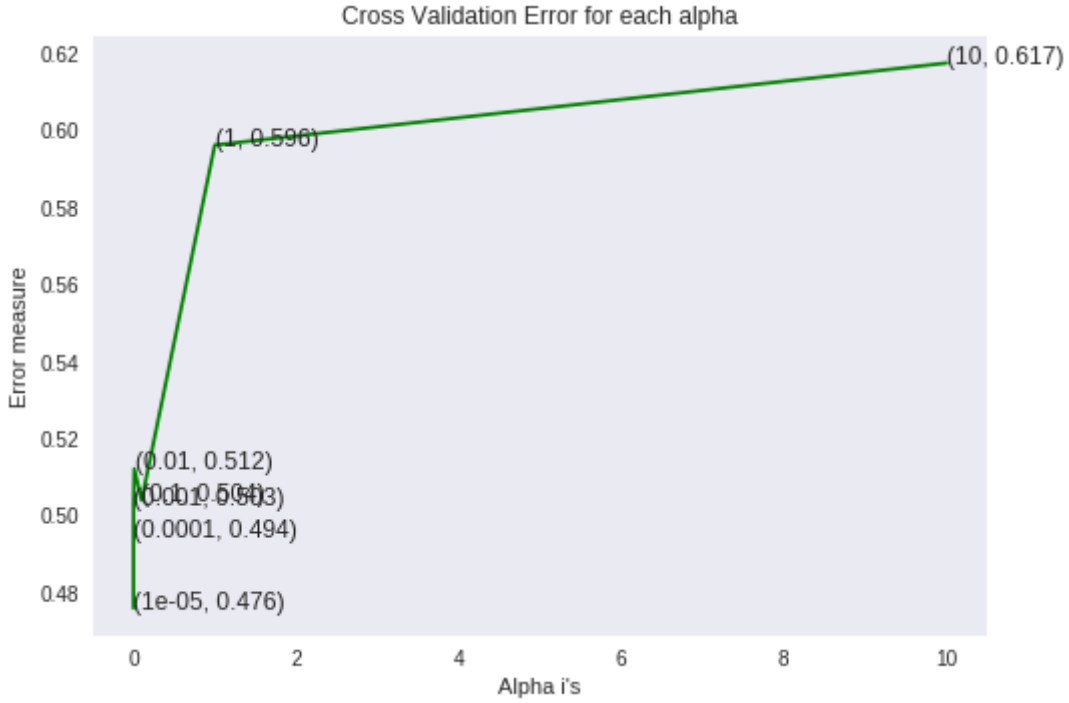
```
X_train = final_data['final_train']
y_train = final_data['y_train']
X_test = final_data['final_test']
y_test = final_data['y_test']
```

In [14]:

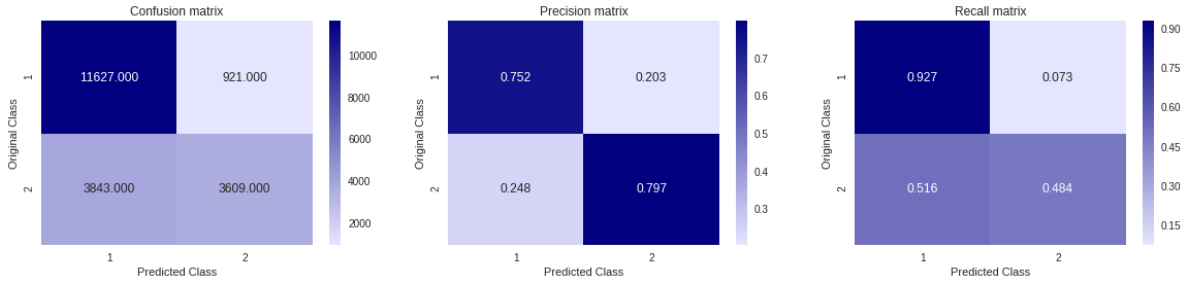
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, 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)
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_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: 0.475897231907471
For values of alpha =  0.0001 The log loss is: 0.4944034560664221
For values of alpha =  0.001 The log loss is: 0.5028007832621897
For values of alpha =  0.01 The log loss is: 0.5120852173340712
For values of alpha =  0.1 The log loss is: 0.5038912747158841
```

```
For values of alpha =  1 The log loss is: 0.5961501764979151
For values of alpha =  10 The log loss is: 0.6174523286821092
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4718495993452656
For values of best alpha =  1e-05 The test log loss is: 0.475897231907471
Total number of data points : 20000
```



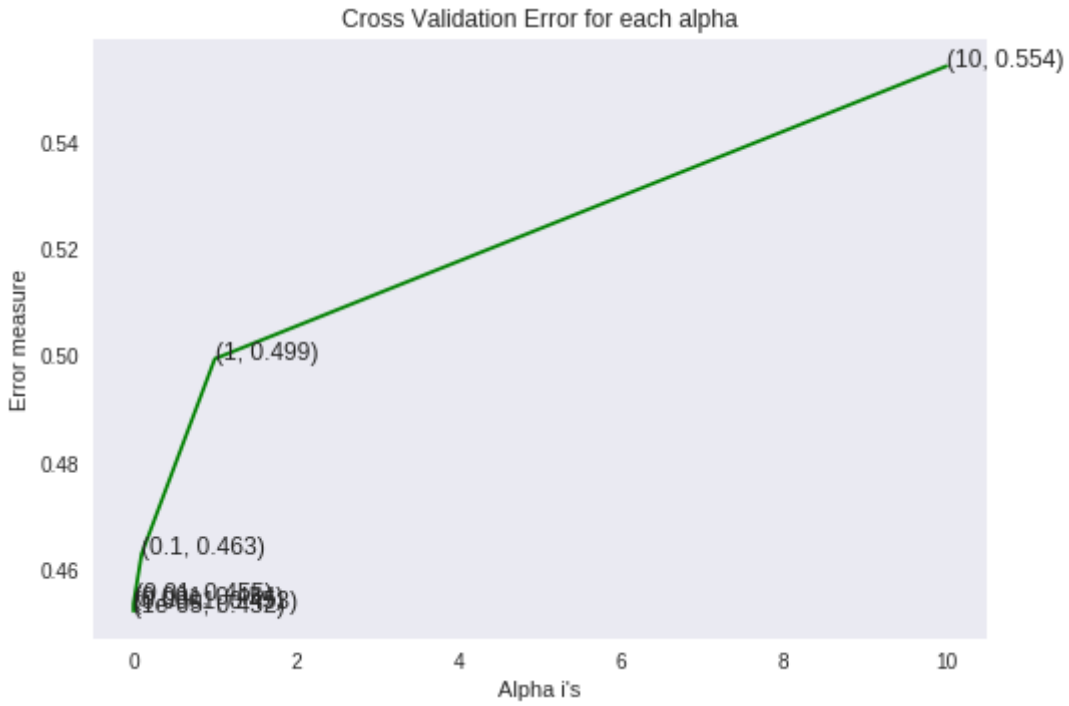
[4.3] Hyperparameter tuning Logistic Regression

In [15]:

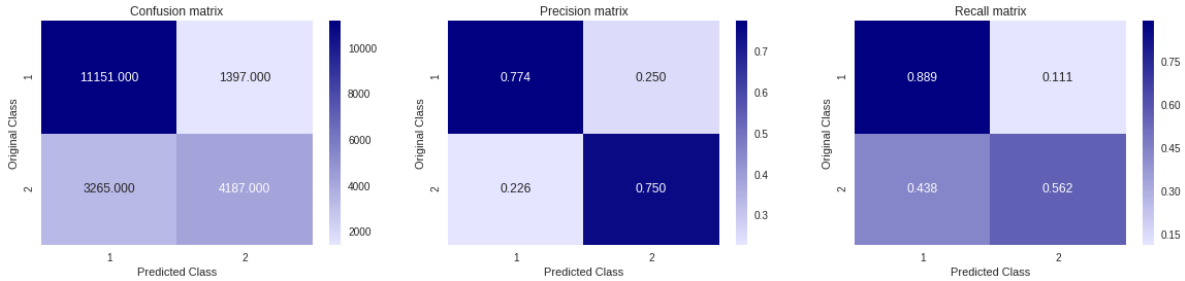
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# 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_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, 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)
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_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: 0.45233213358888724
For values of alpha =  0.0001 The log loss is: 0.45309714506796456
For values of alpha =  0.001 The log loss is: 0.45372808399651177
For values of alpha =  0.01 The log loss is: 0.4546045362121711
For values of alpha =  0.1 The log loss is: 0.4630081230306062
```

```
For values of alpha =  1 The log loss is: 0.4994868097981324
For values of alpha =  10 The log loss is: 0.5541976401888693
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4473179427774883
For values of best alpha =  1e-05 The test log loss is: 0.45233213358888724
Total number of data points : 20000
```



[5] Conclusion

Procedure followed

1. Understood Business problem
2. Mapped it to Machine learning problem
3. Did Exploratory data analysis
   Read data using Pandas and SQLite
   Data understanding like number of data points and their fields
   Found number of unique questions
   Checked for Duplicates and Null values
4. Basic Feature extraction
   Extracted the following basic features like freq_qid1, word_Common, word_share, etc
   Plotted violin plots for some basic features to see their distribution and other properties
5. Text Cleaning
   Removed html tags
   Removed Punctuations
   Performed stemming

Removed Stopwords
Expanded contractions

6. Advanced Feature extraction
   Extracted the following advanced features like cwc_min, cwc_max, fuzz_ratio, token_set_ratio, etc
7. Visualization
   Plotted word clouds, pair plots and violin plots
8. Machine Learning Models
   Hyperparameter tuned XGBoost using RandomizedSearch
   Hyperparameter tuned Linear SVM
   Hyperparameter tuned Logistic Regression
   And printed confusion matrix

In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
x = PrettyTable()
x.field_names = ["Vectorizer", "Algorithm", "Hyperparameters", "Train log loss", "Test log
```

In [19]:

```
x.add_row(["TFIDF", "XGBoost", "max_depth->5 | n_estimators->1000", 0.25, 0.32])
x.add_row(["TFIDF", "Logistic Regression", "alpha->1e-05", 0.44, 0.45])
x.add_row(["TFIDF", "Linear SVM", "alpha->1e-05", 0.47, 0.47])
print(x)
```

```
+------------+---------------------+-----------------------------------+----
------------+--------------+
| Vectorizer |      Algorithm      |          Hyperparameters          | Tra
in log loss | Test log loss |
+------------+---------------------+-----------------------------------+----
------------+--------------+
|   TFIDF    |       XGBoost       | max_depth->5 | n_estimators->1000 |
0.25       |      0.32     |
|   TFIDF    | Logistic Regression |            alpha->1e-05           |
0.44       |      0.45     |
|   TFIDF    |      Linear SVM     |            alpha->1e-05           |
0.47       |      0.47     |
+------------+---------------------+-----------------------------------+----
------------+--------------+
```

Observations

- XGBoost performs better than Logistic regression and Linear SVM lesser train and test log loss.
- XGBoost hyperparameters were tuned using randomized search.