

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 Problem

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Learning 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]:

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

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

3.1 Reading data and basic stats

In [0]:

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

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

Number of data points: 404290

In [0]:

df.head()

Out[8]:

	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 23^{24} i...	0
4	4	9	10	Which one dissolve in water quickly 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         404290 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

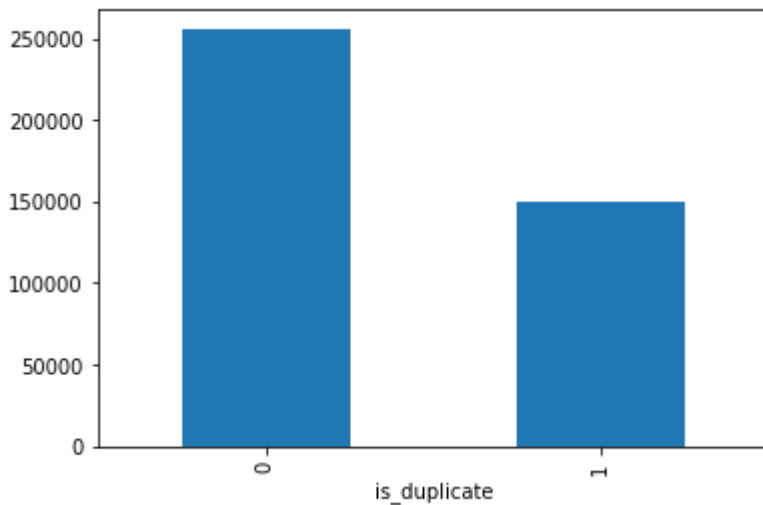
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [0]:

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

Out[10]:

<matplotlib.axes._subplots.AxesSubplot at 0x22b00727d30>



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: {}'.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: {}'.format(max(qids.value_cour
q_vals=qids.value_counts()

q_vals=q_vals.values

```

Total num of Unique Questions are: 537933

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

Max number of times a single question is repeated: 157

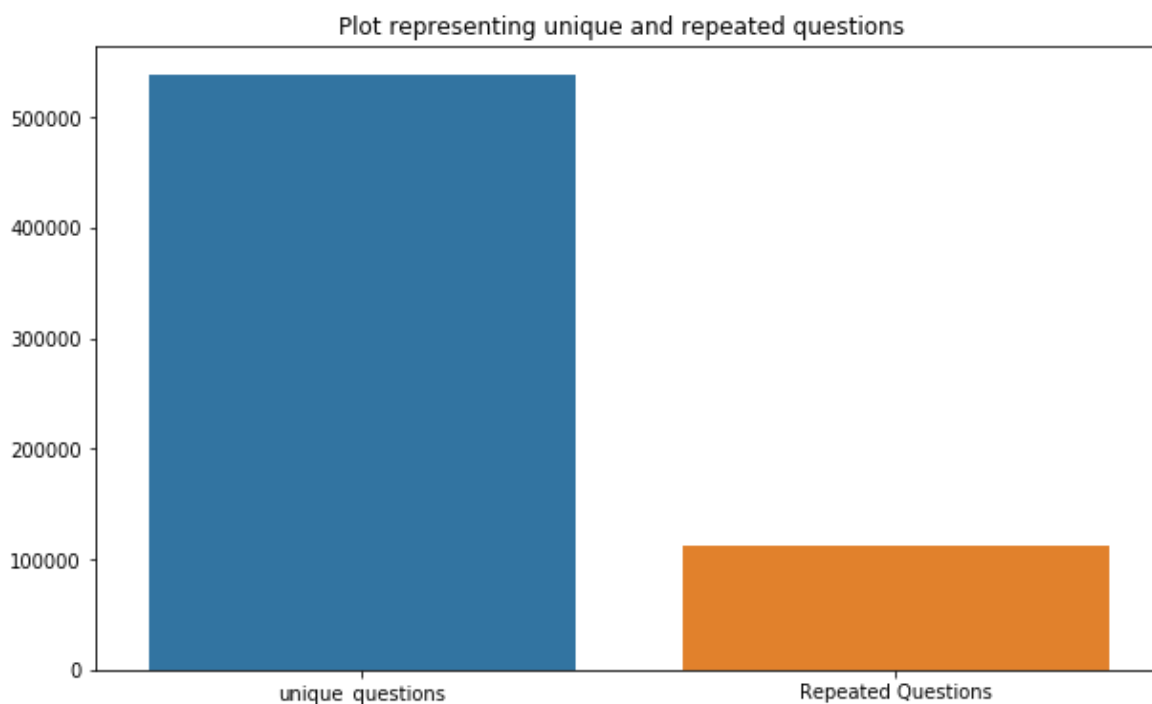
In [0]:

```

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])
```

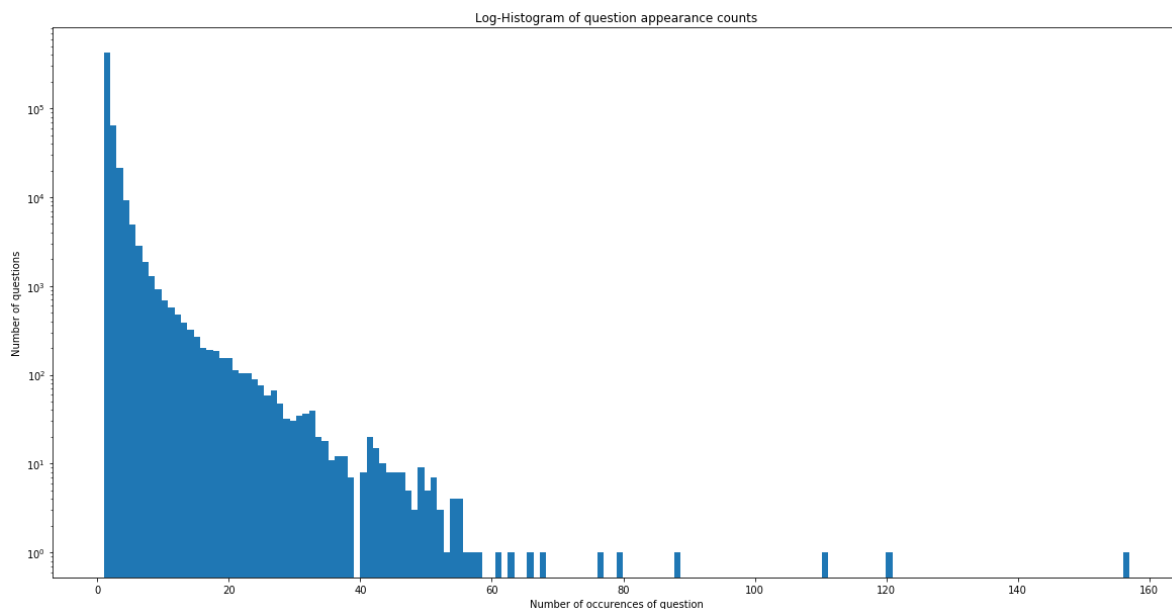
Number of duplicate questions 0

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 occurrences of question')
plt.ylabel('Number of questions')
print ('Maximum number of times a single question is repeated: {}'.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	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

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len
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 23^{24} is divided by 23...	0	1	1	50	65
4	4	9	10	Which one dissolve in water quickly 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].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']== 1].shape[0])
```

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

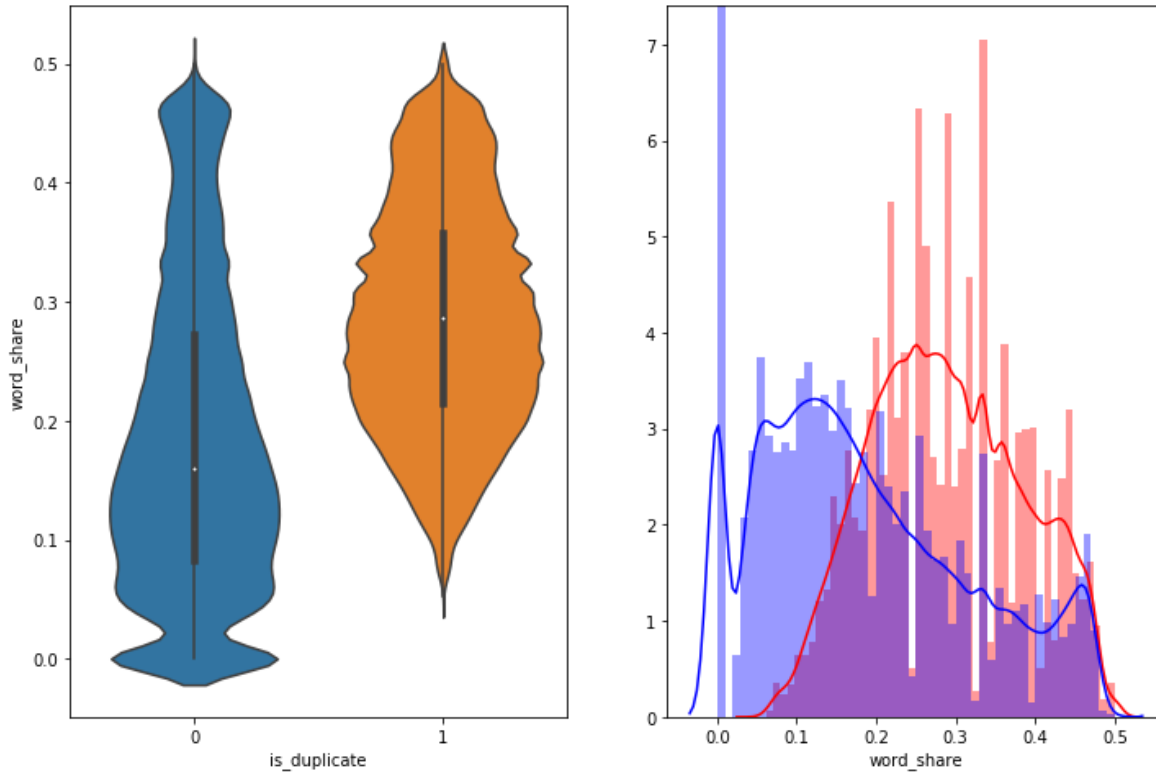
3.3.1.1 Feature: word_share

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)

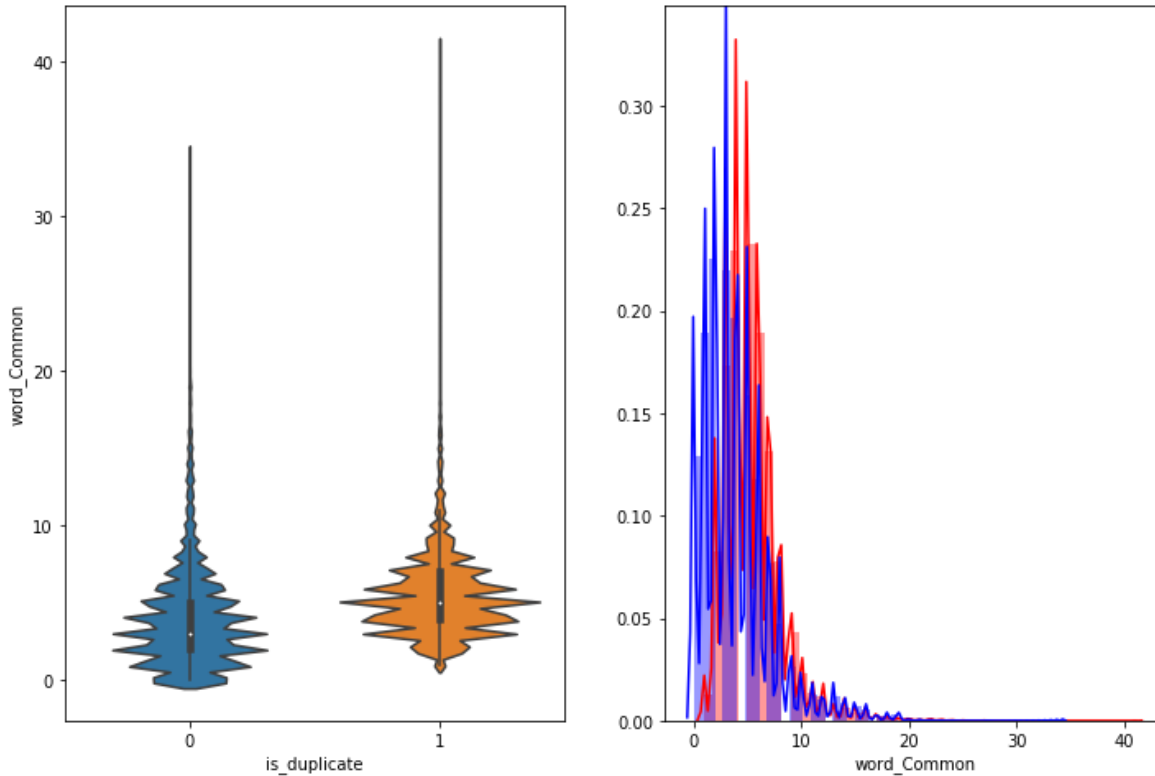
3.3.1.2 Feature: word_Common

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]:

```
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 noteboc")
```

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 length of word count of Q1 and Q2

$$\text{cwc_min} = \text{common_word_count} / (\min(\text{len}(q1_words), \text{len}(q2_words)))$$
- **cwc_max** : Ratio of common_word_count to max length of word count of Q1 and Q2

$$\text{cwc_max} = \text{common_word_count} / (\max(\text{len}(q1_words), \text{len}(q2_words)))$$

- **csc_min** : Ratio of common_stop_count to min length of stop count of Q1 and Q2

$$\text{csc_min} = \text{common_stop_count} / (\min(\text{len}(\text{q1_stops}), \text{len}(\text{q2_stops})))$$
- **csc_max** : Ratio of common_stop_count to max length of stop count of Q1 and Q2

$$\text{csc_max} = \text{common_stop_count} / (\max(\text{len}(\text{q1_stops}), \text{len}(\text{q2_stops})))$$
- **ctc_min** : Ratio of common_token_count to min length of token count of Q1 and Q2

$$\text{ctc_min} = \text{common_token_count} / (\min(\text{len}(\text{q1_tokens}), \text{len}(\text{q2_tokens})))$$
- **ctc_max** : Ratio of common_token_count to max length of token count of Q1 and Q2

$$\text{ctc_max} = \text{common_token_count} / (\max(\text{len}(\text{q1_tokens}), \text{len}(\text{q2_tokens})))$$
- **last_word_eq** : Check if First word of both questions is equal or not

$$\text{last_word_eq} = \text{int}(\text{q1_tokens}[-1] == \text{q2_tokens}[-1])$$
- **first_word_eq** : Check if First word of both questions is equal or not

$$\text{first_word_eq} = \text{int}(\text{q1_tokens}[0] == \text{q2_tokens}[0])$$
- **abs_len_diff** : Abs. length difference

$$\text{abs_len_diff} = \text{abs}(\text{len}(\text{q1_tokens}) - \text{len}(\text{q2_tokens}))$$
- **mean_len** : Average Token Length of both Questions

$$\text{mean_len} = (\text{len}(\text{q1_tokens}) + \text{len}(\text{q2_tokens})) / 2$$
- **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 length of token count of Q1 and Q2

$$\text{longest_substr_ratio} = \text{len}(\text{longest common substring}) / (\min(\text{len}(\text{q1_tokens}), \text{len}(\text{q2_tokens})))$$

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_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)

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

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

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

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

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcs substrings(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["question2"]), axis=1)
# The token sort approach involves tokenizing the string in question, sorting the tokens
# then joining them back into a string We then compare the transformed strings with a set ratio
df["token_sort_ratio"] = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"], x["question2"]), axis=1)
df["fuzz_ratio"] = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), axis=1)
df["fuzz_partial_ratio"] = df.apply(lambda x: fuzz.partial_ratio(x["question1"], x["question2"]), axis=1)
df["longest_substr_ratio"] = df.apply(lambda x: fuzz.longest_common_substring_ratio(x["question1"], x["question2"]), axis=1)
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_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	0.9
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 occurring 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

```
# 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))
```

Word Clouds generated from duplicate pair question's text

```
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()
```

[illegible]

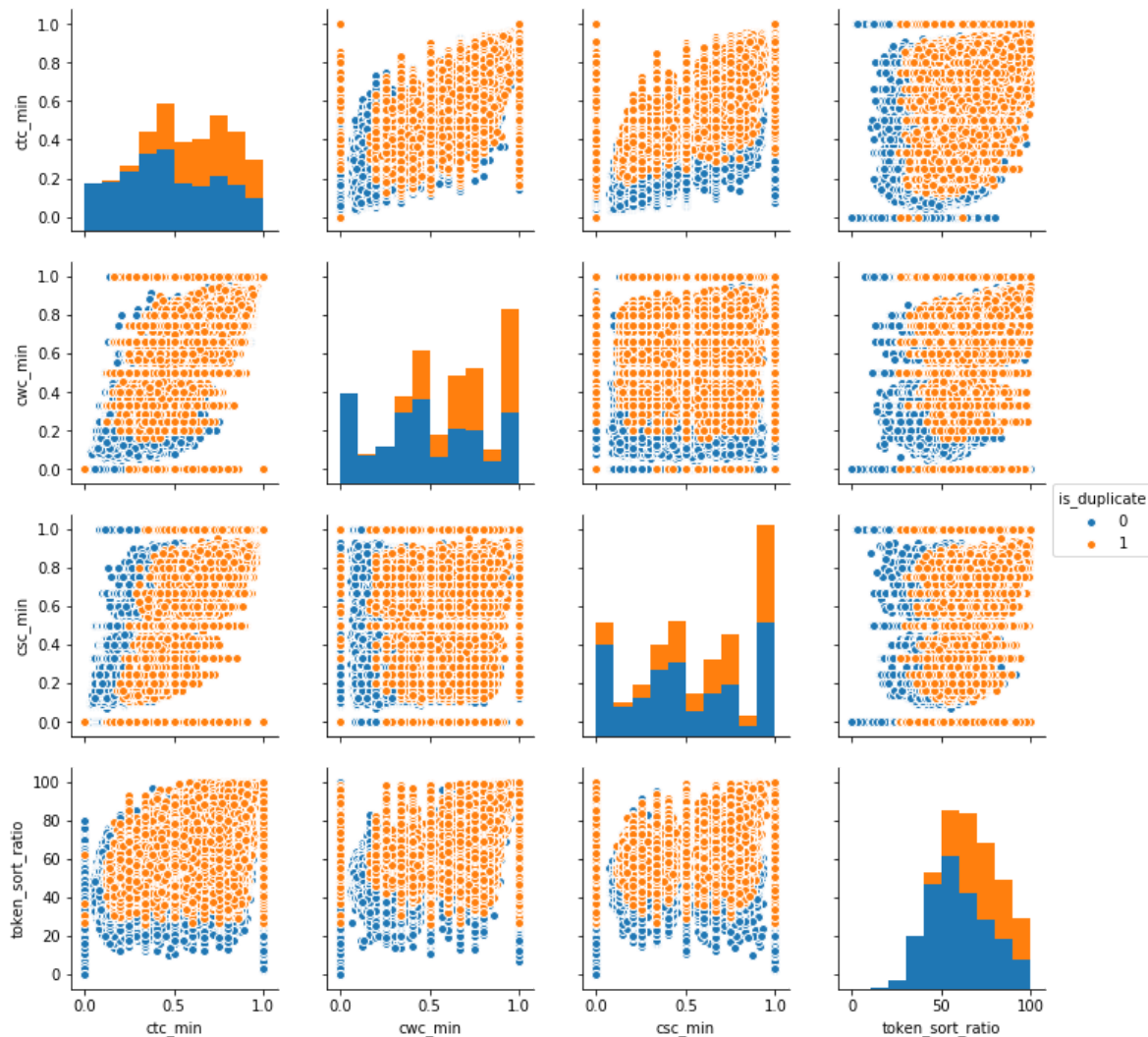
20/54

```
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()
```

[illegible]

In [0]:

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

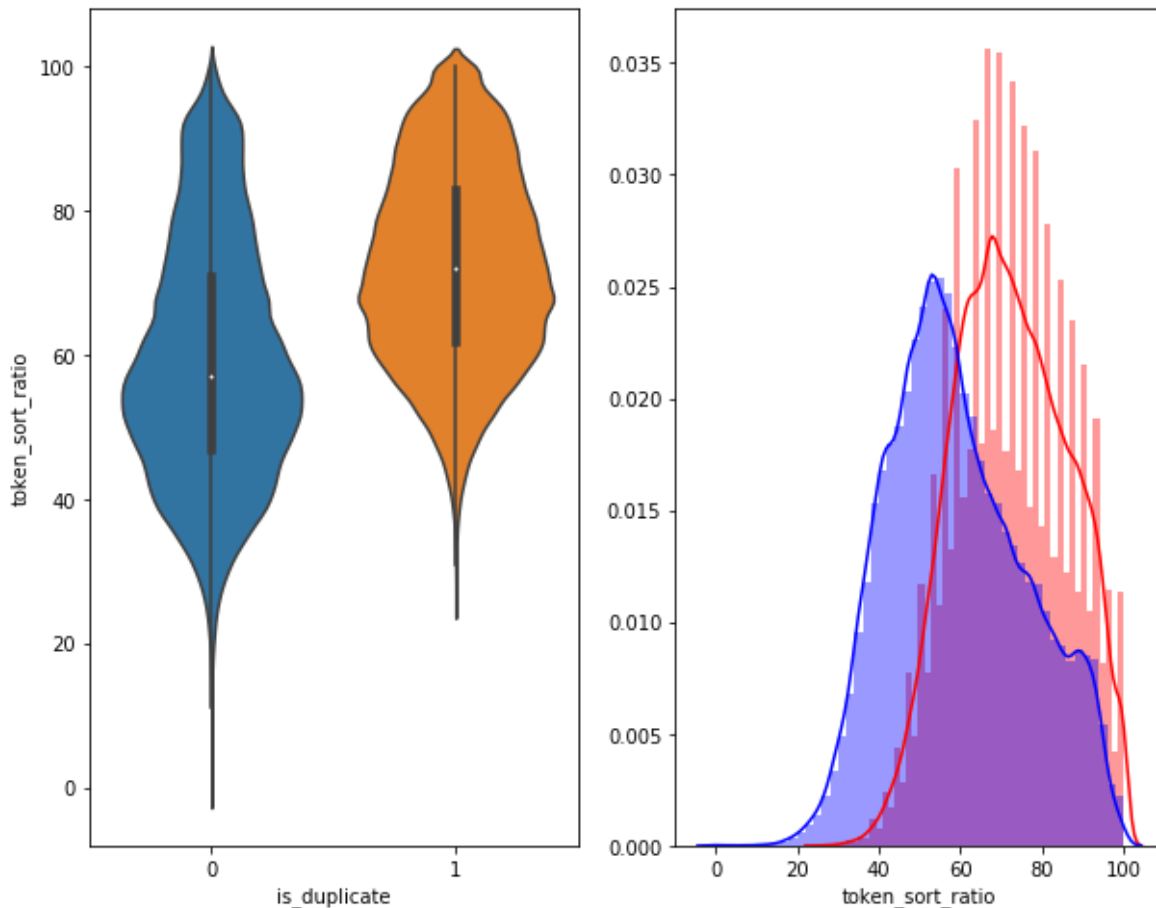


In [0]:

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

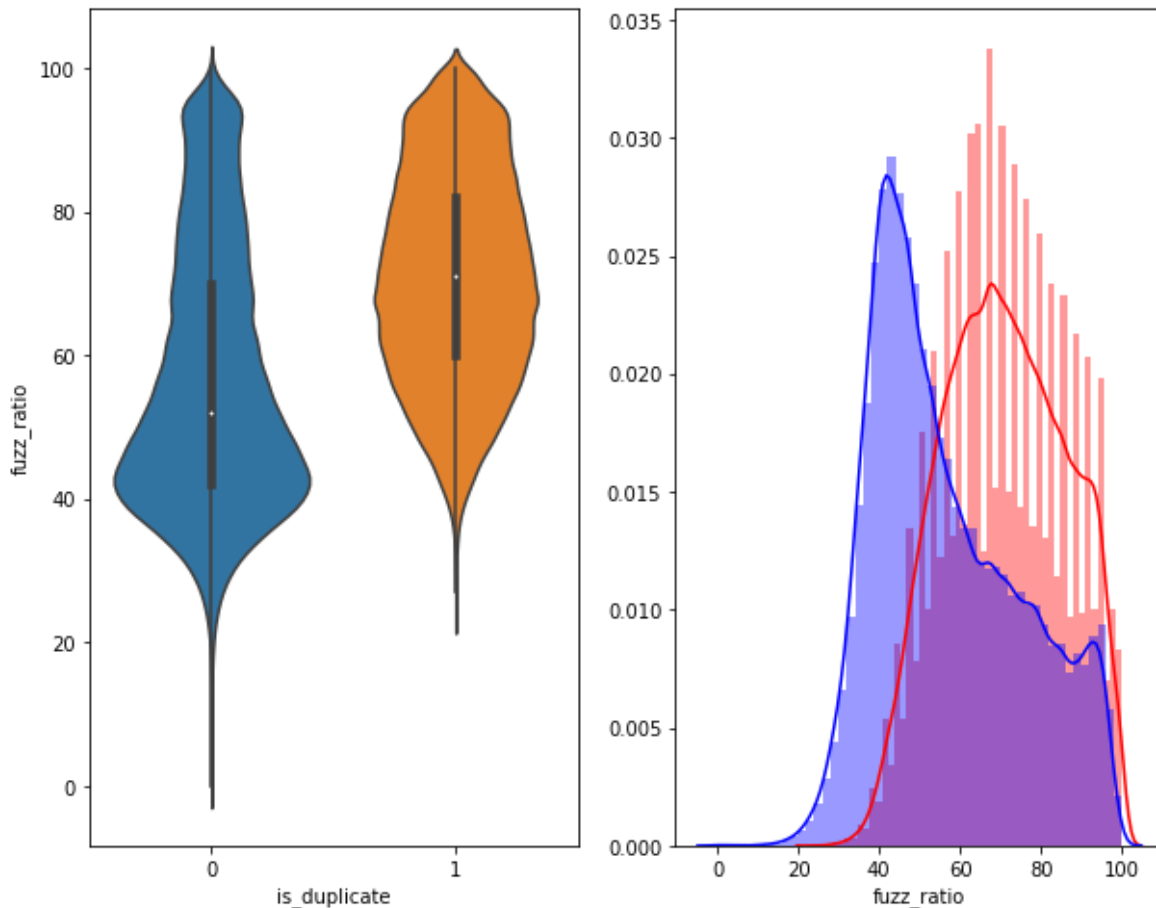


In [0]:

```
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 iterations in 13.099s)
[t-SNE] Iteration 100: error = 70.3858795, gradient norm = 0.0100968 (50 iterations in 9.067s)
[t-SNE] Iteration 150: error = 68.6138382, gradient norm = 0.0058392 (50 iterations in 9.602s)
[t-SNE] Iteration 200: error = 67.7700119, gradient norm = 0.0036596 (50 iterations in 9.121s)
[t-SNE] Iteration 250: error = 67.2725067, gradient norm = 0.0034962 (50 iterations in 11.305s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.272507
[t-SNE] Iteration 300: error = 1.7737305, gradient norm = 0.0011918 (50 iterations in 8.289s)
[t-SNE] Iteration 350: error = 1.3720417, gradient norm = 0.0004822 (50 iterations in 10.526s)
[t-SNE] Iteration 400: error = 1.2039998, gradient norm = 0.0002768 (50 iterations in 9.600s)
[t-SNE] Iteration 450: error = 1.1133438, gradient norm = 0.0001881 (50 iterations in 11.827s)
[t-SNE] Iteration 500: error = 1.0579143, gradient norm = 0.0001434 (50 iterations in 8.941s)
[t-SNE] Iteration 550: error = 1.0221983, gradient norm = 0.0001164 (50 iterations in 11.092s)
[t-SNE] Iteration 600: error = 0.9987167, gradient norm = 0.0001039 (50 iterations in 11.467s)
[t-SNE] Iteration 650: error = 0.9831534, gradient norm = 0.0000938 (50 iterations in 11.799s)
[t-SNE] Iteration 700: error = 0.9722011, gradient norm = 0.0000858 (50 iterations in 12.028s)
[t-SNE] Iteration 750: error = 0.9643636, gradient norm = 0.0000799 (50 iterations in 12.120s)
[t-SNE] Iteration 800: error = 0.9584482, gradient norm = 0.0000785 (50 iterations in 11.867s)
[t-SNE] Iteration 850: error = 0.9538348, gradient norm = 0.0000739 (50 iterations in 11.461s)
[t-SNE] Iteration 900: error = 0.9496906, gradient norm = 0.0000712 (50 iterations in 11.023s)
[t-SNE] Iteration 950: error = 0.9463405, gradient norm = 0.0000673 (50 iterations in 11.023s)
```

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]:

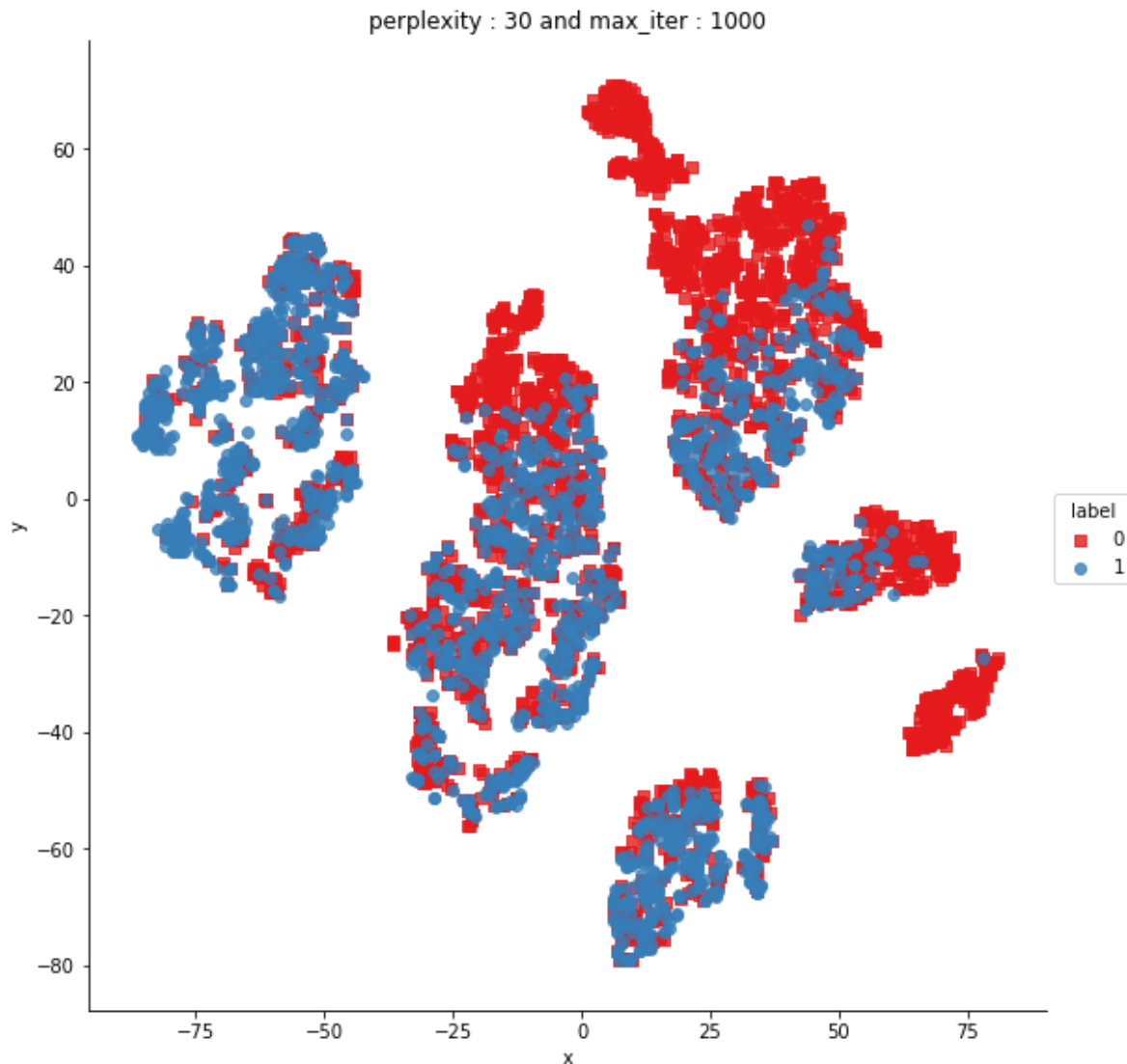
```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] , 'label':y})
```

```
# 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()
```



In [0]:

```
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 0.010s...
[t-SNE] Computed neighbors for 5000 samples in 0.935s...
[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 0.363s
[t-SNE] Iteration 50: error = 77.7944183, gradient norm = 0.1014017 (50 iterations in 34.931s)
[t-SNE] Iteration 100: error = 69.2682266, gradient norm = 0.0248657 (50 iterations in 15.147s)
[t-SNE] Iteration 150: error = 67.7877655, gradient norm = 0.0150941 (50 iterations in 13.761s)
[t-SNE] Iteration 200: error = 67.1991119, gradient norm = 0.0126559 (50 iterations in 13.425s)
[t-SNE] Iteration 250: error = 66.8560715, gradient norm = 0.0074975 (50 iterations in 12.904s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.856071
[t-SNE] Iteration 300: error = 1.2356015, gradient norm = 0.0007033 (50 iterations in 13.302s)
[t-SNE] Iteration 350: error = 0.9948602, gradient norm = 0.0001997 (50 iterations in 18.898s)
[t-SNE] Iteration 400: error = 0.9168936, gradient norm = 0.0001430 (50 iterations in 13.397s)
[t-SNE] Iteration 450: error = 0.8863022, gradient norm = 0.0000975 (50 iterations in 16.379s)
[t-SNE] Iteration 500: error = 0.8681002, gradient norm = 0.0000854 (50 iterations in 17.791s)
[t-SNE] Iteration 550: error = 0.8564141, gradient norm = 0.0000694 (50 iterations in 17.060s)
[t-SNE] Iteration 600: error = 0.8470711, gradient norm = 0.0000640 (50 iterations in 15.454s)
[t-SNE] Iteration 650: error = 0.8389117, gradient norm = 0.0000561 (50 iterations in 17.562s)
[t-SNE] Iteration 700: error = 0.8325295, gradient norm = 0.0000529 (50 iterations in 13.443s)
[t-SNE] Iteration 750: error = 0.8268463, gradient norm = 0.0000528 (50 iterations in 17.981s)
[t-SNE] Iteration 800: error = 0.8219477, gradient norm = 0.0000477 (50 iterations in 17.448s)
[t-SNE] Iteration 850: error = 0.8180174, gradient norm = 0.0000490 (50 iterations in 18.376s)
[t-SNE] Iteration 900: error = 0.8150476, gradient norm = 0.0000456 (50 iterations in 17.778s)
```

```
[t-SNE] Iteration 950: error = 0.8122067, gradient norm = 0.0000472 (50 iterations in 16.983s)
[t-SNE] Iteration 1000: error = 0.8095787, gradient norm = 0.0000489 (50 iterations in 18.581s)
[t-SNE] Error after 1000 iterations: 0.809579
```

In [0]:

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

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

3.6 Featurizing text data with tfidf 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

# extract 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("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[61]:

	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 23^{24} 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.shape

Out[65]:

(404290, 6)

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

- After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores.

- In [0]:

```
100%|██████████████████████████████████████████████████████████████████████████|  
█ 404290/404290 [2:13:51<00:00, 50.34it/s]
```

In [0]:

```
100%|██████████████████████████████████████████████████████████████████████████|  
█ | 404290/404290 [1:47:52<00:00, 62.46it/s]
```

In [0]:

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

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [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)
```

In [0]:

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

Out[9]:

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

In [0]:

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

Out[10]:

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

In [0]:

```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[11]:

	0	1	2	3	4	5	6
0	121.929927	100.083900	72.497894	115.641800	-48.370870	34.619058	-172.057787
1	-78.070939	54.843781	82.738482	98.191872	-51.234859	55.013510	-39.140730
2	-5.355015	73.671810	14.376365	104.130241	1.433537	35.229116	-148.519385
3	5.778359	-34.712038	48.999631	59.699204	40.661263	-41.658731	-36.808594
4	51.138220	38.587312	123.639488	53.333041	-47.062739	37.356212	-298.722753

5 rows × 384 columns

In [0]:

```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[12]:

	0	1	2	3	4	5	6	7
0	125.983301	95.636485	42.114702	95.449980	-37.386295	39.400078	-148.116070	-1
1	-106.871904	80.290331	79.066297	59.302092	-42.175328	117.616655	-144.364237	-
2	7.072875	15.513378	1.846914	85.937583	-33.808811	94.702337	-122.256856	-
3	39.421531	44.136989	-24.010929	85.265863	-0.339022	-9.323137	-60.499651	-
4	31.950101	62.854106	1.778164	36.218768	-45.130875	66.674880	-106.342341	-

5 rows × 384 columns

In [0]:

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

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v dataframe : 384
Number of features in question2 w2v dataframe : 384
Number of features in final dataframe : 794
```


In [0]:

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

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
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier

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

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

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [0]:

```
#Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('final_features.csv', names=['Unnamed: 0', 'id', 'is_duplicate', 'c
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
```

In [0]:

```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the databse:")
    tables = table_names.fetchall()
    print(tables[0][0])
    return(len(tables))
```

In [0]:

```
read_db = 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

Tables in the databse:
data

In [0]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)

        # for selecting random points
        data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn_r)
        conn_r.commit()
        conn_r.close()
```

In [0]:

```
data.index = range(len(data))
```

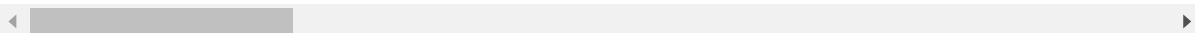
In [0]:

```
data.head()
```

Out[24]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min
0	0.999950002499875	0.999950002499875	0.66664444518516	0.499987500312492	0.799987500312492
1	0.66664444518516	0.66664444518516	0.749981250468738	0.749981250468738	0.719981250468738
2	0.199996000079998	0.199996000079998	0.66664444518516	0.285710204139941	0.375710204139941
3	0.0	0.0	0.33332222259258	0.199996000079998	0.166666666666667
4	0.499987500312492	0.499987500312492	0.749981250468738	0.499991666805553	0.624991666805553

5 rows × 794 columns



In [0]:

```
data.columns
```

Out[59]:

```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
      'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
      ...,
      '374_y', '375_y', '376_y', '377_y', '378_y', '379_y', '380_y', '381_y',
      '382_y', '383_y'],
      dtype='object', length=794)
```

In [0]:

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

In [0]:

```
for idx, item in enumerate(y_true.values):
    if item == "is_duplicate":
        print(item, idx)

print(len(y_true.values))
```

```
is_duplicate 18781
100000
```

4.2 Converting strings to numerics

In [0]:

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

In [0]:

```
type(data['cwc_min'][1])
```

Out[33]:

```
numpy.float64
```

In [0]:

```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
# y_true = list(map(int, y_true.values))
y_labels = []
for i in y_true.values:
    if i == "is_duplicate":
        i = "1"
        y_labels.append(int(i))
    else:
        y_labels.append(int(i))
```

In [0]:

```
y_true = y_labels
```

4.3 Random train test split(70:30)

In [0]:

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

In [0]:

```
nlp_features = pd.read_csv("nlp_features_train.csv", encoding='latin-1')
```

In [0]:

```
features.head()  
print(features.columns[29])  
print(features.columns[412])  
print(features.columns[413])  
print(features.columns[796])  
print(features.shape[1])
```

```
0_x  
383_x  
0_y  
383_y  
797
```

In [0]:

```
nlp_features.head()
```

Out[6]:

	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	2	5	6	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	0.2
3	3	7	8	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	0.0
4	4	9	10	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.6

5 rows × 21 columns



In [0]:

```
features_new = features.copy()
features_new.drop(features_new.iloc[:, 29:797], inplace = True, axis = 1)
features_new.shape
```

Out[7]:

(404290, 29)

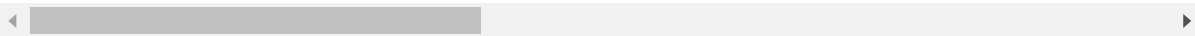
In [0]:

```
features_new.head()
```

Out[8]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_m
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.7857
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.4666
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.2857
3	3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
4	4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.3076

5 rows × 29 columns



In [0]:

```
final_csv = pd.concat([features_new, nlp_features['question1'], nlp_features['question2']],
final_csv.shape
final_csv.to_csv("final_csv.csv", index = False)
```

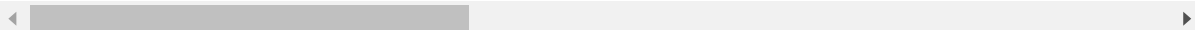
In [0]:

```
final_csv.head()
```

Out[10]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_m
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.7857
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.4666
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.2857
3	3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
4	4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.3076

5 rows × 31 columns



In [0]:

```
y_label = final_csv['is_duplicate']
final_csv = final_csv.drop(['Unnamed: 0', 'id', 'is_duplicate'], axis = 1)
```


In [0]:

```
final_csv.columns
```

Out[12]:

```
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', 'freq_qid1', 'freq_qid2',
      'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
      'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'question1',
      'question2'],
      dtype='object')
```

In [0]:

```
X_train,X_test, y_train, y_test = train_test_split(final_csv, y_label, stratify=y_label, te
```

In [0]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
print(X_train.columns)
```

(283003, 28)

(121287, 28)

(283003,)

(121287,)

```
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', 'freq_qid1', 'freq_qid2',
      'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
      'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'question1',
      'question2'],
      dtype='object')
```

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_q1 = TfidfVectorizer(min_df=10)
X_train_tfidf_q1 = tfidf_vectorizer_q1.fit_transform(X_train['question1'].values.astype(str))
X_test_tfidf_q1 = tfidf_vectorizer_q1.transform(X_test['question1'].values.astype(str))
print("Shape of X_train TFIDF question1: ", X_train_tfidf_q1.shape)
print("Shape of X_test TFIDF question1: ", X_test_tfidf_q1.shape)
```

Shape of X_train TFIDF question1: (283003, 12257)

Shape of X_test TFIDF question1: (121287, 12257)

In [0]:

```
tfidf_vectorizer_q2 = TfidfVectorizer(min_df=10)
X_train_tfidf_q2 = tfidf_vectorizer_q2.fit_transform(X_train['question2'].values.astype(str))
X_test_tfidf_q2 = tfidf_vectorizer_q2.transform(X_test['question2'].values.astype(str))
print("Shape of X_train TFIDF question2: ", X_train_tfidf_q2.shape)
print("Shape of X_test TFIDF question2: ", X_test_tfidf_q2.shape)
```

Shape of X_train TFIDF question2: (283003, 11830)

Shape of X_test TFIDF question2: (121287, 11830)

Merging Questions/ Text Features

In [0]:

```
from scipy import sparse
from scipy.sparse import coo_matrix, hstack, vstack
```

In [0]:

```
X_train_questions = hstack([X_train_tfidf_q1, X_train_tfidf_q2])
X_test_questions = hstack([X_test_tfidf_q1, X_test_tfidf_q2])
print(X_train_questions.shape)
print(X_test_questions.shape)
```

(283003, 24087)

(121287, 24087)

Merging Numerical/ Categorical Features

In [0]:

```
X_train_numeric = X_train.drop(['question1', 'question2'], axis = 1)
X_test_numeric = X_test.drop(['question1', 'question2'], axis = 1)
print(X_train_numeric.columns)
print(X_test_numeric.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', 'freq_qid1', 'freq_qid
2',
      'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
      'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
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', 'freq_qid1', 'freq_qid
2',
      'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
      'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

In [0]:

```
X_train_numeric = sparse.csr_matrix(X_train_numeric)
X_test_numeric = sparse.csr_matrix(X_test_numeric)
print(type(X_train_numeric))
print(type(X_test_numeric))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

Merging all the features

In [0]:

```
X_train_tfidf_final = hstack([X_train_questions, X_train_numeric])
X_test_tfidf_final = hstack([X_test_questions, X_test_numeric])
print(X_train_tfidf_final.shape)
print(X_test_tfidf_final.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(283003, 24113)
(121287, 24113)
(283003,)
(121287,)
```

In [0]:

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

```
Number of data points in train data : (283003, 24113)
Number of data points in test data : (121287, 24113)
```

In [0]:

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

```
----- Distribution of output variable in train data -----
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
----- Distribution of output variable in train data -----
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

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

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

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

    plt.show()
```

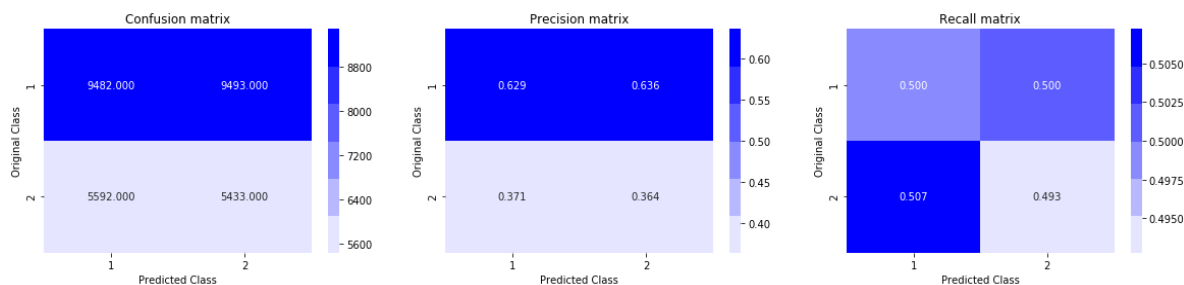
4.4 Building a random model (Finding worst-case log-loss)

In [0]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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 0.887242646958



5. Assignments

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.

5.1 Logistic Regression with hyperparameter tuning

In [0]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn
# -----
# 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 Descent
# predict(X) Predict class labels for samples in X.

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

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

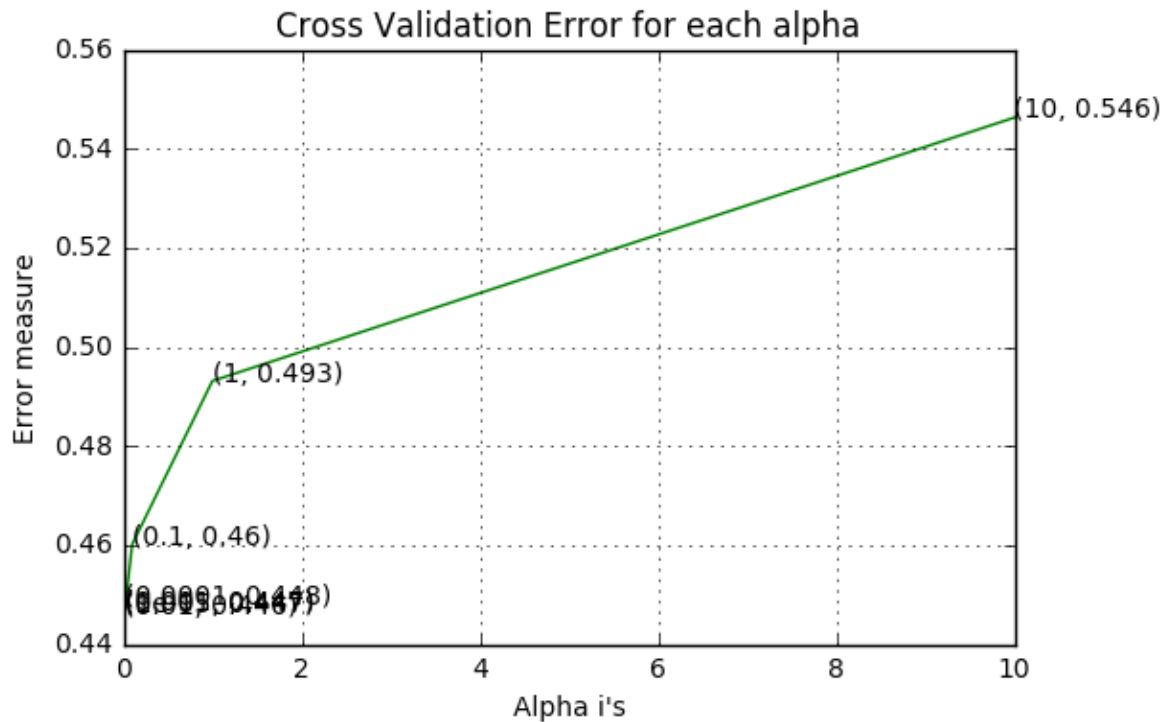
For values of alpha = 1e-05 The log loss is: 0.44664603869768454
For values of alpha = 0.0001 The log loss is: 0.44814400587493874
For values of alpha = 0.001 The log loss is: 0.4467881110180478
For values of alpha = 0.01 The log loss is: 0.4460320883205229
For values of alpha = 0.1 The log loss is: 0.46016358232893434
For values of alpha = 1 The log loss is: 0.4931104989213217
For values of alpha = 10 The log loss is: 0.5462568206091304
```

In [0]:

```

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()

```



In [0]:

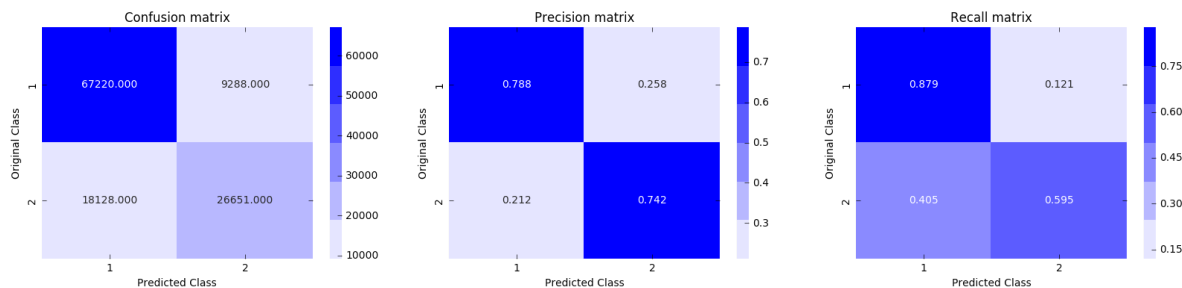
```
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42, n_j
clf.fit(X_train_tfidf_final, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_tfidf_final, y_train)

predict_y = sig_clf.predict_proba(X_train_tfidf_final)
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_tfidf_final)
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 best alpha = 0.01 The train log loss is: 0.44636587942069866

For values of best alpha = 0.01 The test log loss is: 0.4460320883205229

Total number of data points : 121287



5.2 Linear SVM with hyperparameter tuning

In [0]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn
# -----
# 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 Descent
# predict(X) Predict class labels for samples in X.

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

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

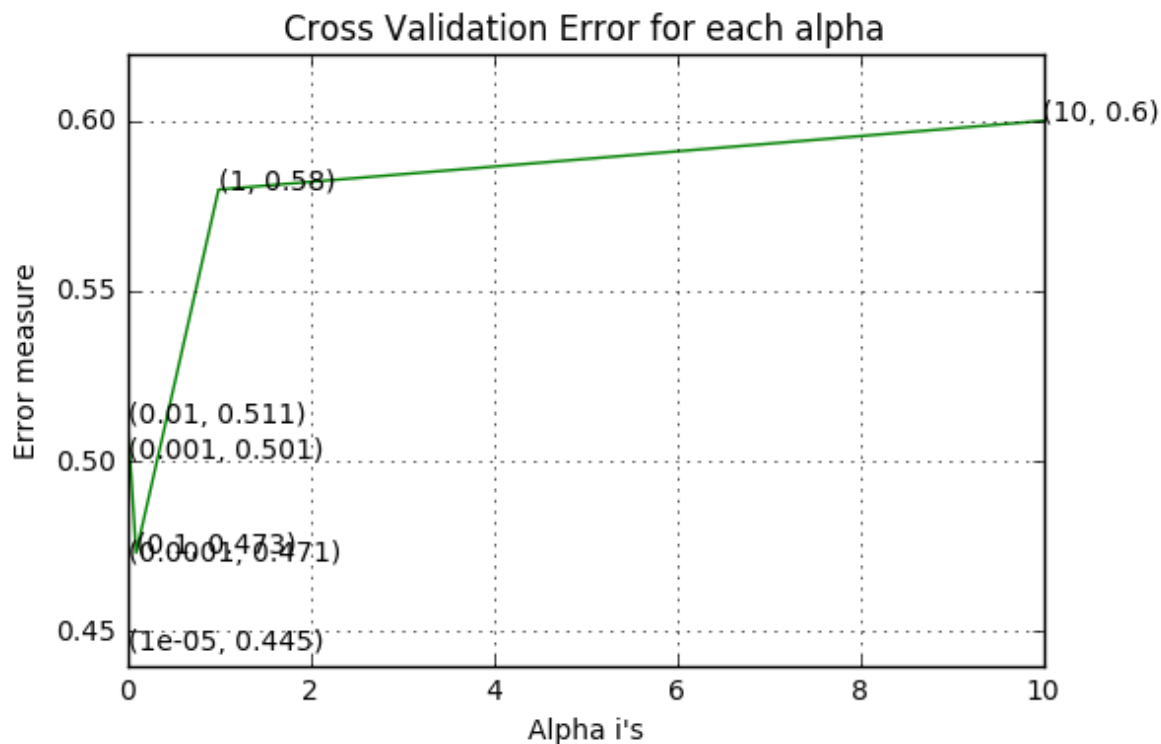
For values of alpha = 1e-05 The log loss is: 0.4445377609032383
For values of alpha = 0.0001 The log loss is: 0.4708785625977367
For values of alpha = 0.001 The log loss is: 0.501063511994122
For values of alpha = 0.01 The log loss is: 0.5112582929511381
For values of alpha = 0.1 The log loss is: 0.4730618183687803
For values of alpha = 1 The log loss is: 0.5799081277687792
For values of alpha = 10 The log loss is: 0.6001121984788736
```

In [0]:

```

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()

```



In [0]:

```

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

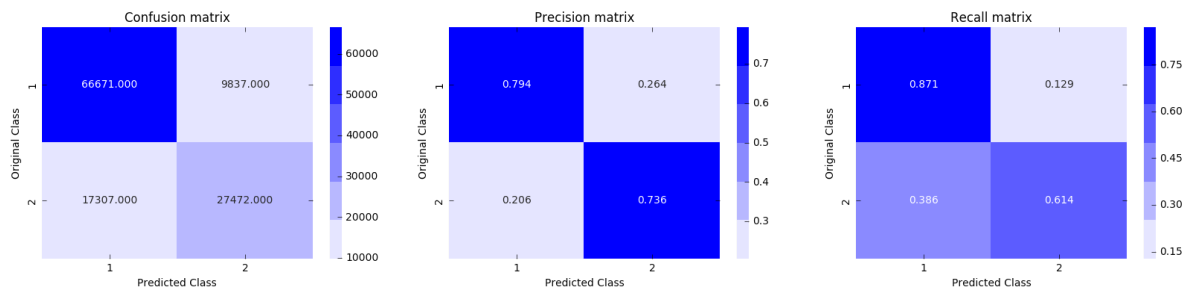
predict_y = sig_clf.predict_proba(X_train_tfidf_final)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(X_test_tfidf_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_
predicted_y = 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 best alpha = 1e-05 The train log loss is: 0.44524083991526325

For values of best alpha = 1e-05 The test log loss is: 0.4445377609032383

Total number of data points : 121287



5.3 XGBoost

Hyperparameter tuning Using RandomSearch

In [0]:

```

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

```

In [0]:

```

param_grid = {'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10], 'n_estimators': [5, 10, 50, 100, 2

```

In [0]:

```
print(X_train_tfidf_final.shape)
print(X_test_tfidf_final.shape)

X_train_rcv = vstack((X_train_tfidf_final, X_test_tfidf_final))
y_train_rcv = pd.concat([y_train, y_test], axis = 0)

print(X_train_rcv.shape)
print(y_train_rcv.shape)
```

```
(283003, 24113)
(121287, 24113)
(404290, 24113)
(404290,)
```

In [0]:

```
random_search_xgb = RandomizedSearchCV(XGBClassifier(), param_grid, scoring = 'neg_log_loss',
                                       return_train_score = True, verbose = 2, n_jobs = -1)

random_search_xgb.fit(X_train_rcv, y_train_rcv)
```

In [0]:

```
with open("random_search_xgb.pkl", "wb") as file:
    pickle.dump(random_search_xgb, file)
```

In [0]:

```
print("Best Parameters: ", random_search_xgb.best_params_)
print("Best Estimator: ", random_search_xgb.best_estimator_)
bst = random_search_xgb.best_estimator_
```

```
Best Parameters: {'max_depth': 9, 'n_estimators': 200}
Best Estimator: XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
                           colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
                           max_delta_step=0, max_depth=9, min_child_weight=1, missing=None,
                           n_estimators=200, n_jobs=1, nthread=None,
                           objective='binary:logistic', random_state=0, reg_alpha=0,
                           reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
                           subsample=1, verbosity=1)
```

In [0]:

```
predict_y = bst.predict_proba(X_train_tfidf_final)
print("The train log loss is:", log_loss(y_train, predict_y, labels=bst.classes_, eps=1e-15))

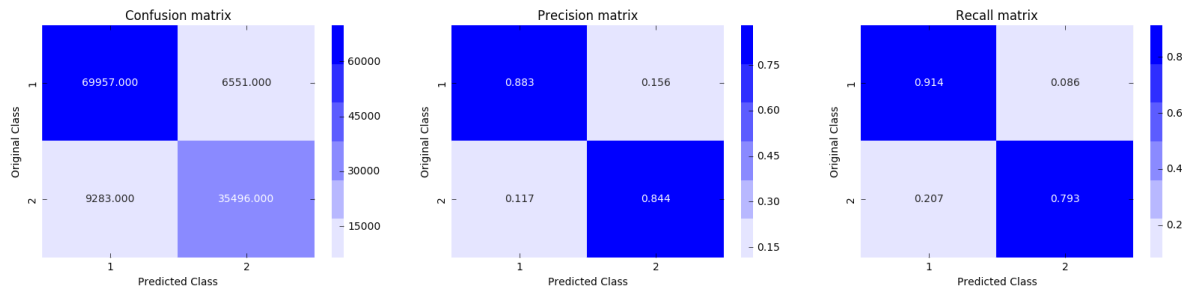
predict_y = bst.predict_proba(X_test_tfidf_final)
print("The test log loss is:", log_loss(y_test, predict_y, labels=bst.classes_, eps=1e-15))

The train log loss is: 0.28114260876053837
The test log loss is: 0.2820839729945564
```

In [0]:

```
# predicted_y = np.array(predict_y>0.5,dtype=int)
predicted_y = np.argmax(predict_y, axis = 1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

Total number of data points : 121287



In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
x = PrettyTable()
x.field_names = ["Algorithm", "loss"]

x.add_row(["Logistic Regression", 0.4460])
x.add_row(["Linear SVM", 0.4445])
x.add_row(["XGBoost", 0.2820])
print(x)
```

```
+-----+-----+
| Algorithm | loss |
+-----+-----+
| Logistic Regression | 0.446 |
| Linear SVM | 0.4445 |
| XGBoost | 0.282 |
+-----+-----+
```

Procedure Followed

- 1) Loaded and merged CSV files having Question1, Question2 and nlp features.
- 2) Split the data into Train and test ratio of 70:30.
- 3) Vectorized questions (text) using TFIDF and made a sparse matrix of Numeric features.
- 4) Merged the the above two using Horizontal Stacking (hstack).
- 5) Trained a Logistic Regression model with Hyperparameter tuning.
- 6) Trained a Linear SVM model with Hyperparameter tuning.
- 7) Trained XGBoost model with Hyperparameter tuning using RandomSearchCV.

