# Business Problem

## 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.

## 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)

## 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.

# Machine Learning Probelm

## Data

### 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

### 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"
```

## Mapping the real world problem to an ML problem

### 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.

### 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

# 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
```

## 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 [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       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:

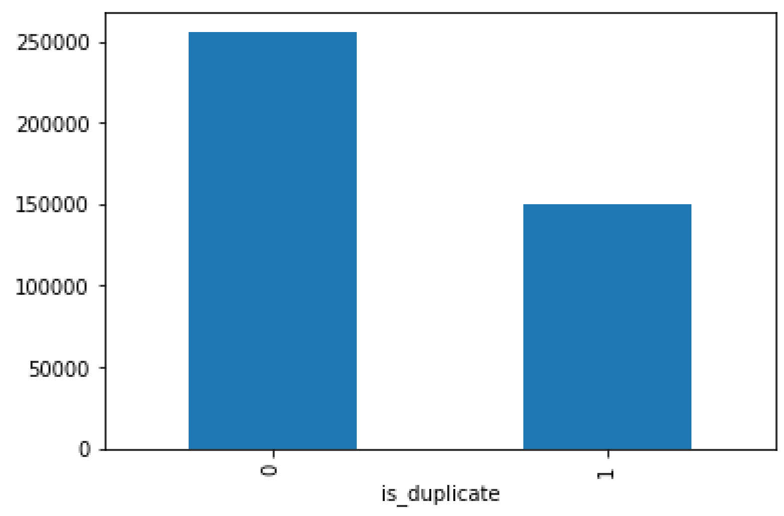
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

### 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("Number of question pairs available for training is",len(df))
```

```
Number of question pairs available for training is 404290
```

In [0]:
```
print("Percentage of question pairs that are not similar (is_duplicate = 0) is",100 - round(df['is_duplicate'].mean()*100, 2))
print("Percentage of question pairs that are similar (is_duplicate = 1) is",round(df['is_duplicate'].mean()*100, 2))
```

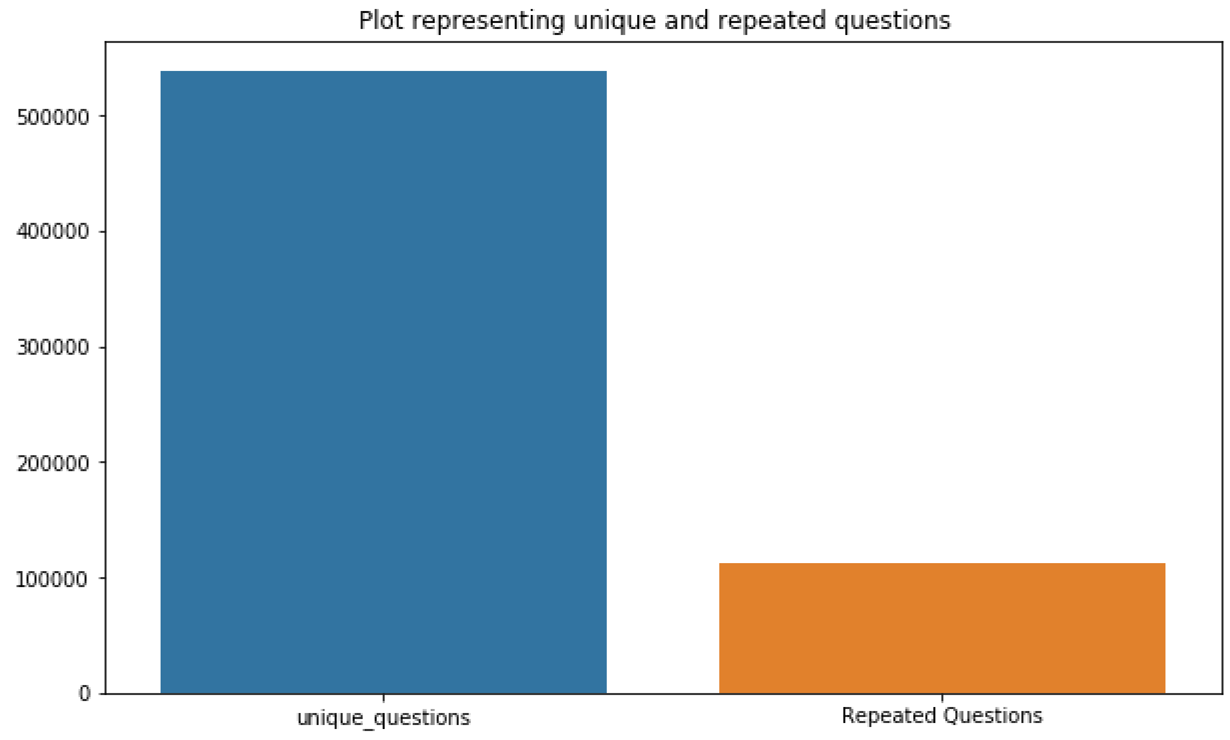
```
Percentage of question pairs that are not similar (is_duplicate = 0) is 63.08%
Percentage of question pairs that are similar (is_duplicate = 1) is 36.92%
```

### 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_morethan_onetime,qs_morethan_onetime/unique_qs*100))

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

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()
```




### Checking for Duplicates

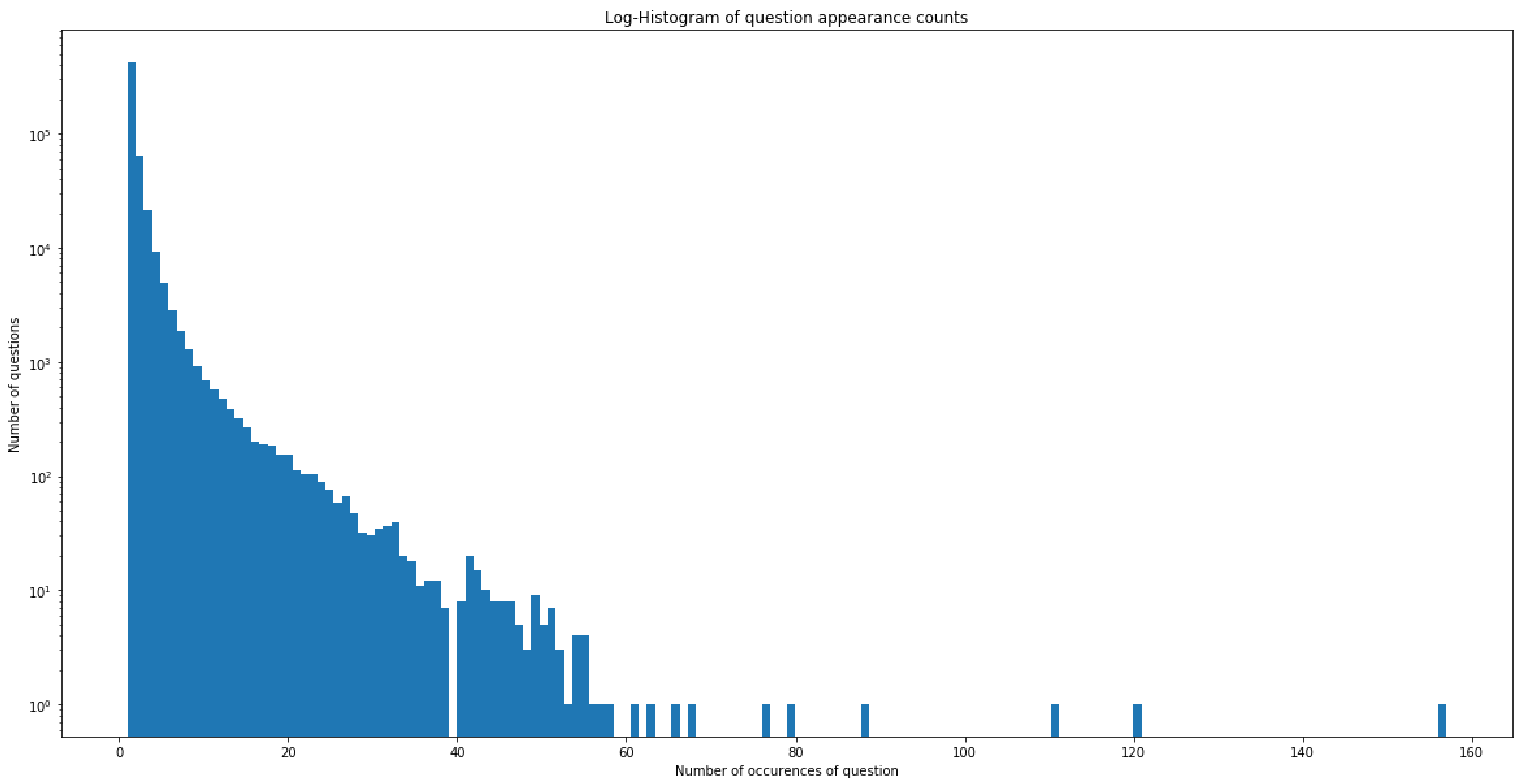
In [0]:
```
#checking whether there are any repeated pair of questions

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

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

```
Number of duplicate questions 0
```

### 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_counts())))
```

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




### 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: []
```

# 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 | q1_n_words | q2_n_words | word_Common | word_Total | word_share | freq_q1+q2 | freq_q1-q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 14 | 12 | 10.0 | 23.0 | 0.434783 | 2 | 0 |
| **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 | 8 | 13 | 4.0 | 20.0 | 0.200000 | 5 | 3 |
| **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 | 14 | 10 | 4.0 | 24.0 | 0.166667 | 2 | 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 | 1 | 1 | 50 | 65 | 11 | 9 | 0.0 | 19.0 | 0.000000 | 2 | 0 |
| **4** | 4 | 9 | 10 | Which one dissolve in water quikly sugar, salt... | Which fish would survive in salt water? | 0 | 3 | 1 | 76 | 39 | 13 | 7 | 2.0 | 20.0 | 0.100000 | 4 | 2 |

### Analysis of some of the extracted features

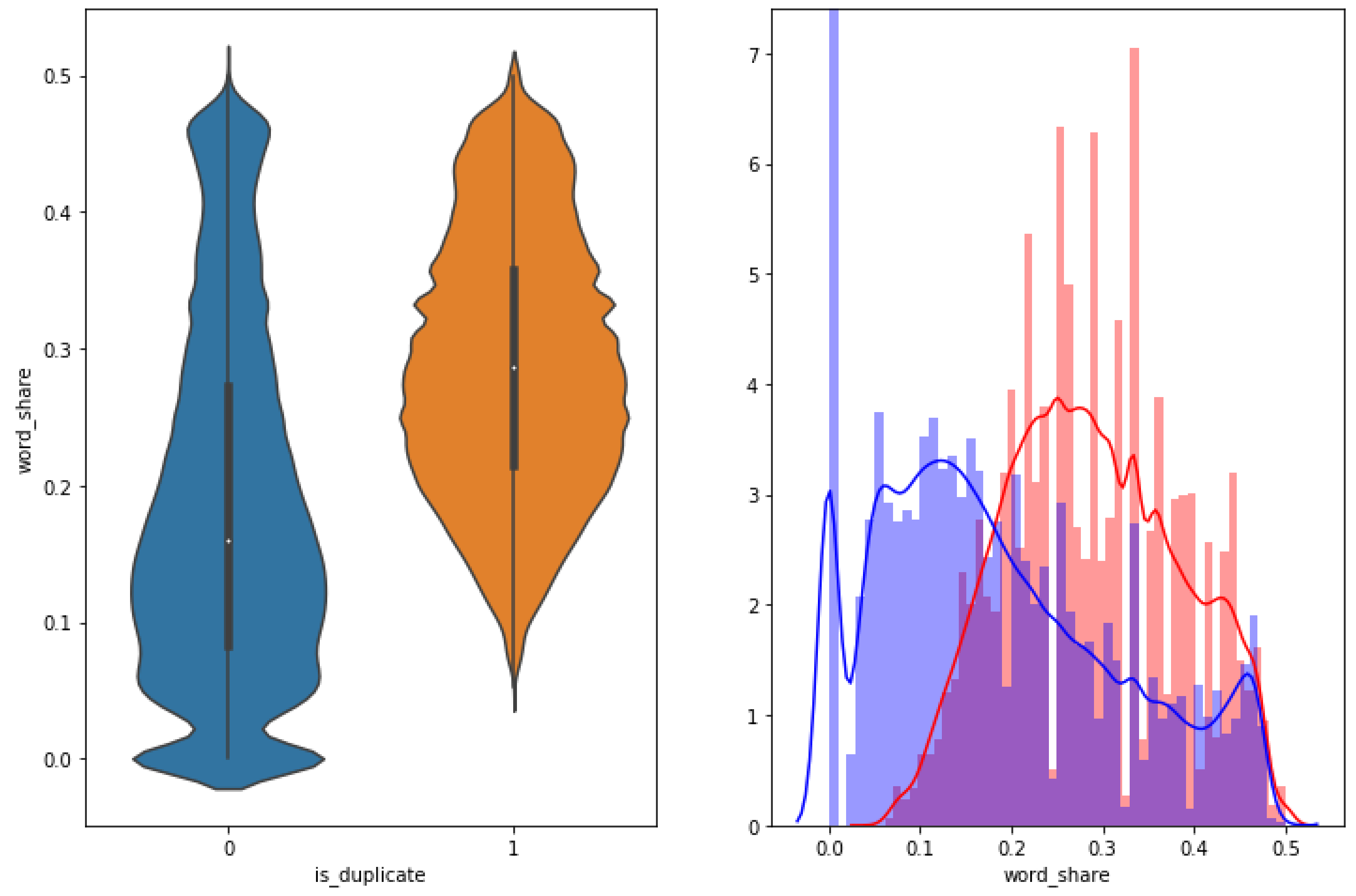
- Here are some questions have only one single words.

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
```

**Feature: word_share**

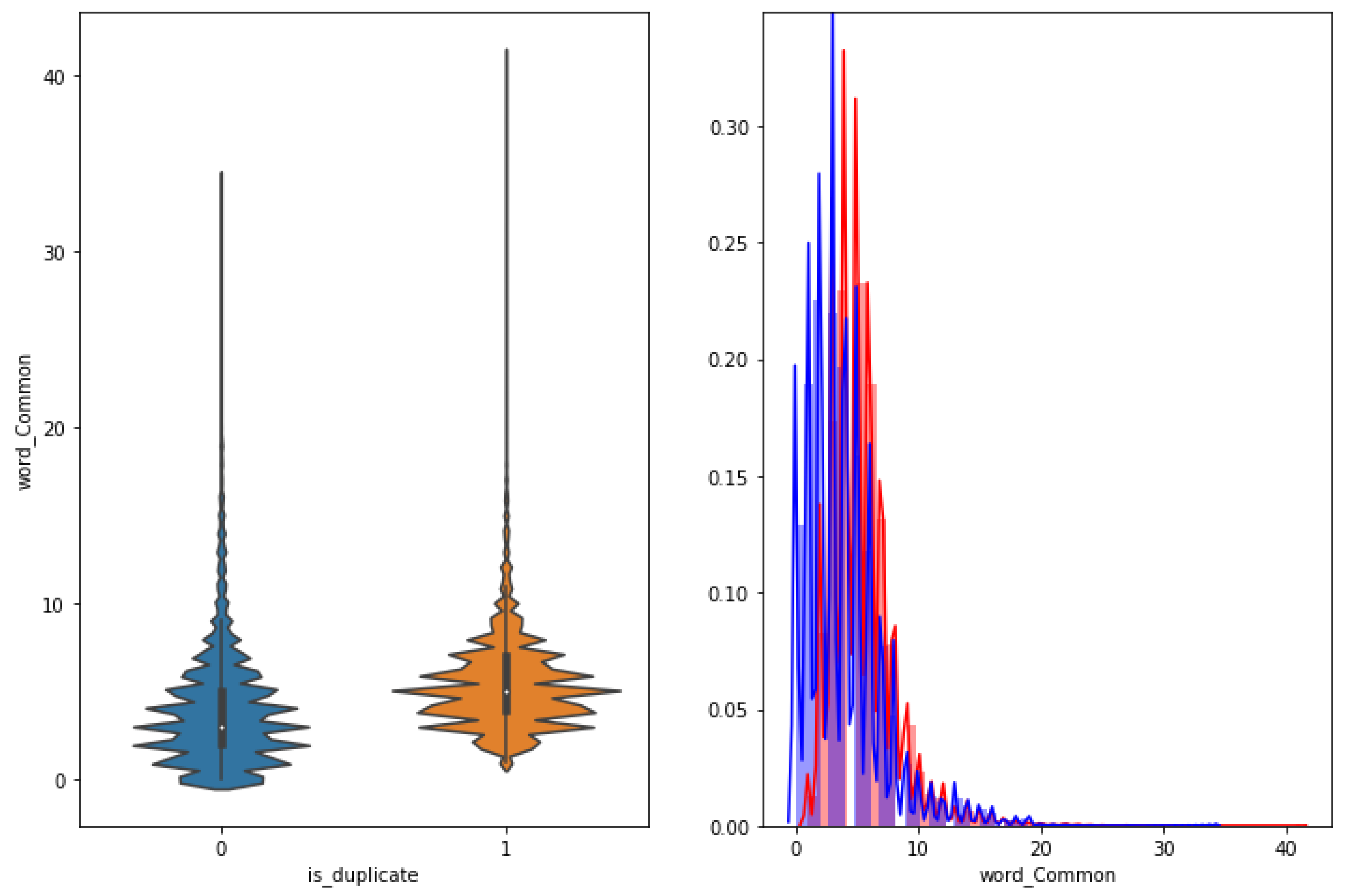
In [0]:
```
plt.figure(figsize=(12, 8))

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)

**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

## 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-byte-0x9c
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
```

In [0]:
```
df.head(2)
```

Out[8]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common | word_Total | word_share | freq_q1+q2 | freq_q1-q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 14 | 12 | 10.0 | 23.0 | 0.434783 | 2 | 0 |
| **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 | 8 | 13 | 4.0 | 20.0 | 0.200000 | 5 | 3 |

## 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").replace("can't", "can not")\
                           .replace("n't", " not").replace("what's", "what is").replace("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").replace("'s", " own")\
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar ")\
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    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

## 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 First word of both questions is equal or not
  last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])

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

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

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

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

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

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

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

- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

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

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

    print("token features...")

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

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

In [0]:
```python
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 | ... | ctc_max | last_word_eq | first_word_eq | abs_len_diff | mean_len | token_set_ratio | token_sort_ratio | fuzz_ratio | fuzz_partial_ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | what is the step by step guide to invest in sh... | what is the step by step guide to invest in sh... | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | ... | 0.785709 | 0.0 | 1.0 | 2.0 | 13.0 | 100 | 93 | 93 | 10 |
| **1** | 1 | 3 | 4 | what is the story of kohinoor koh i noor dia... | what would happen if the indian government sto... | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | ... | 0.466664 | 0.0 | 1.0 | 5.0 | 12.5 | 86 | 63 | 66 | 7 |

2 rows × 21 columns

### Analysis of extracted features

#### 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
```

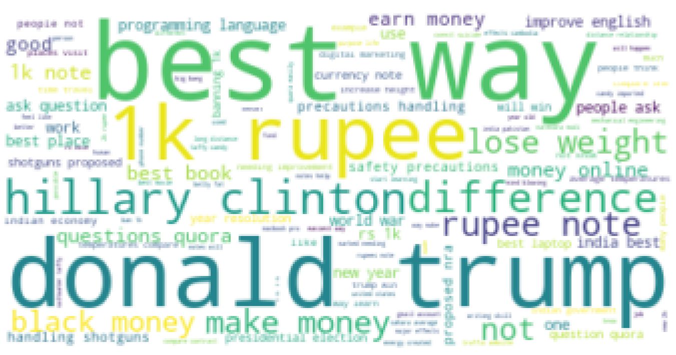
In [0]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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()
```

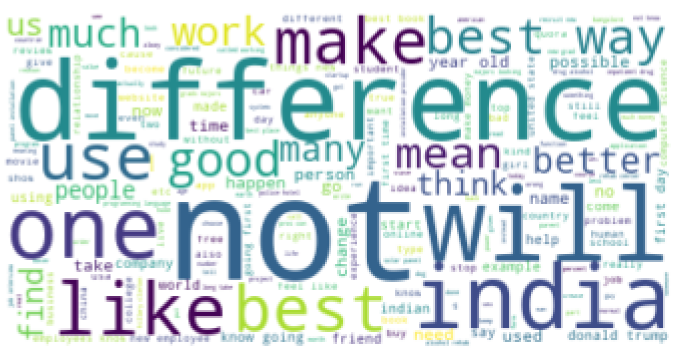
```
Word Cloud for Duplicate Question pairs
```




**Word Clouds generated from non duplicate pair question's text**

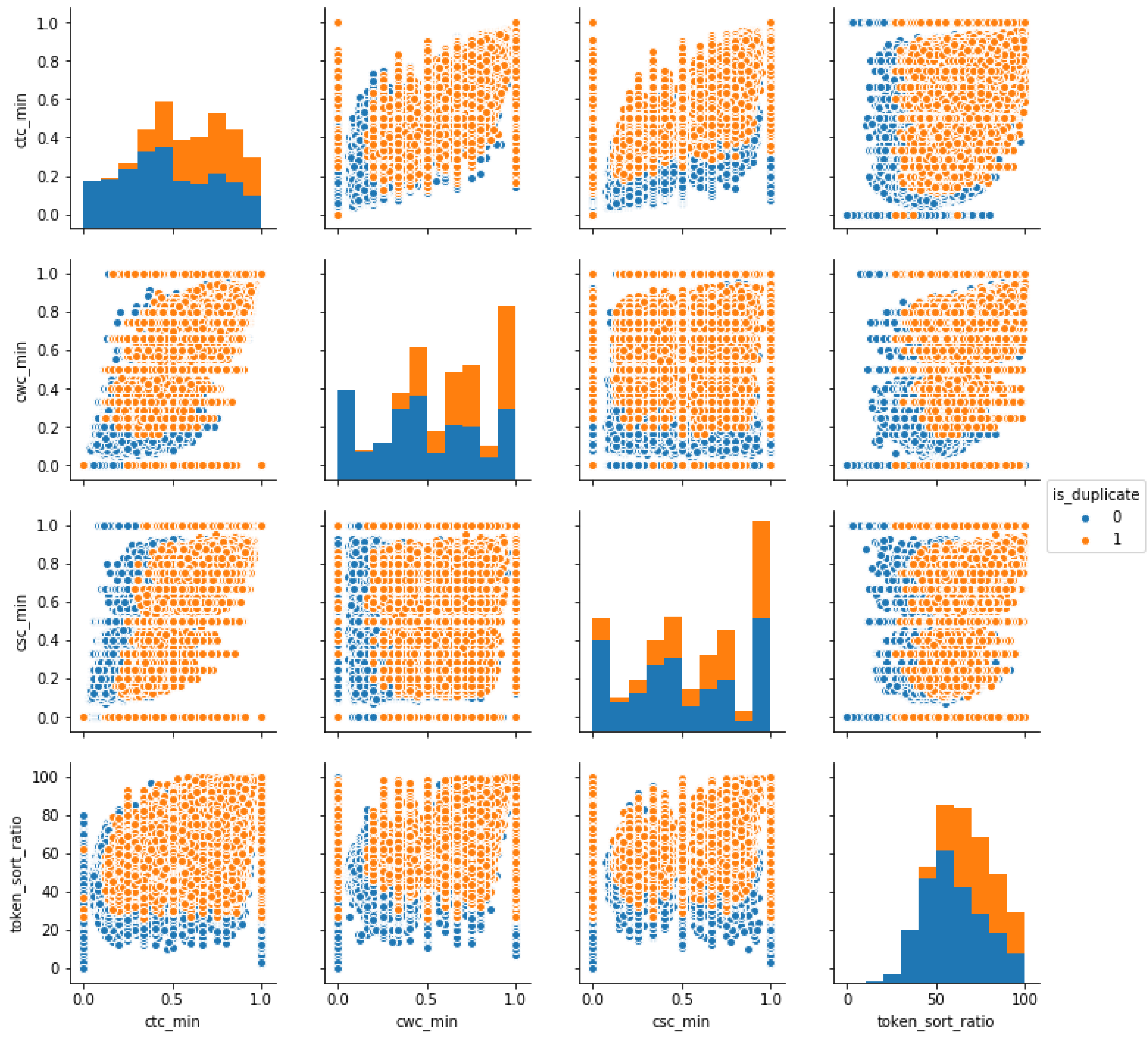
In [0]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

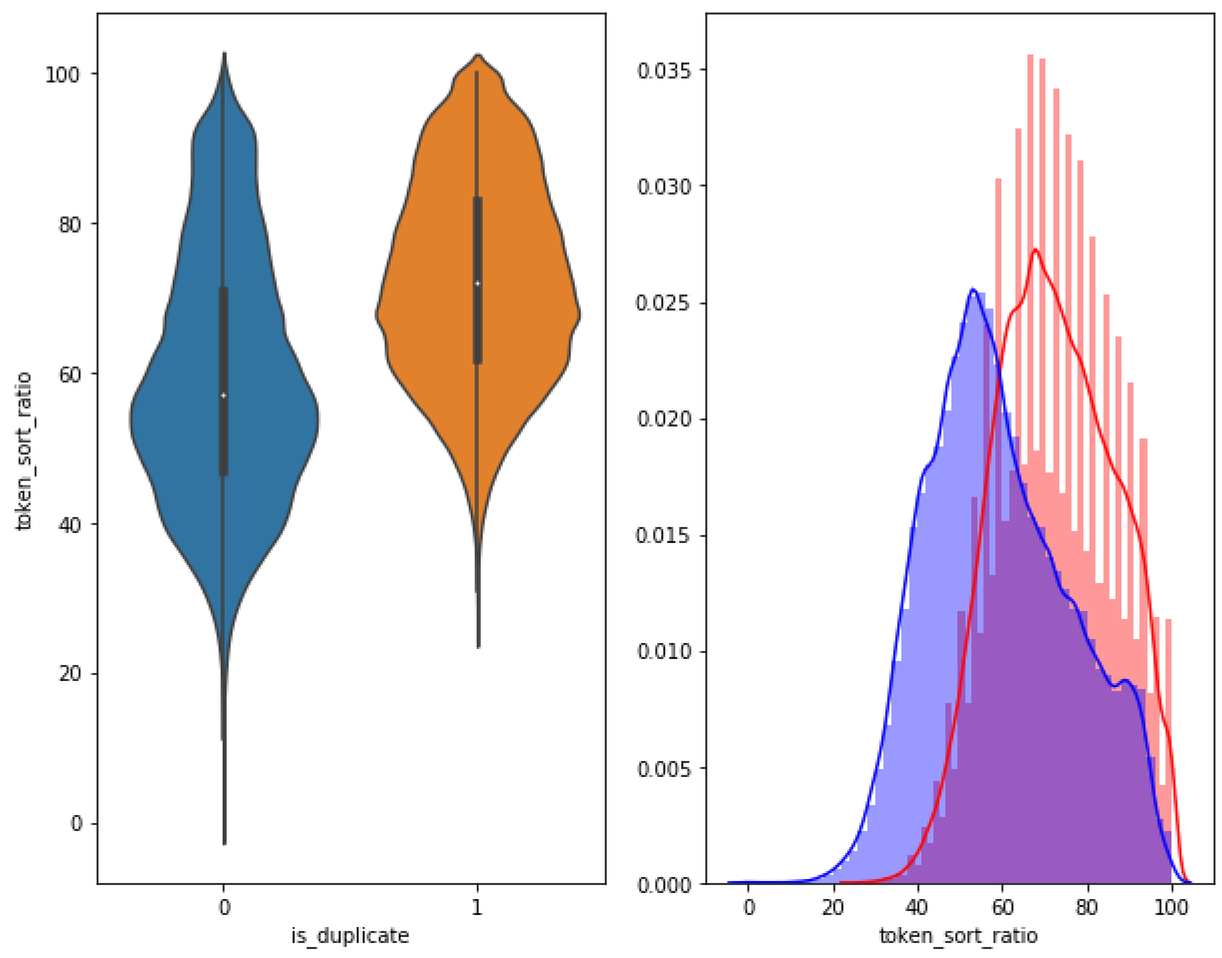



**Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']**

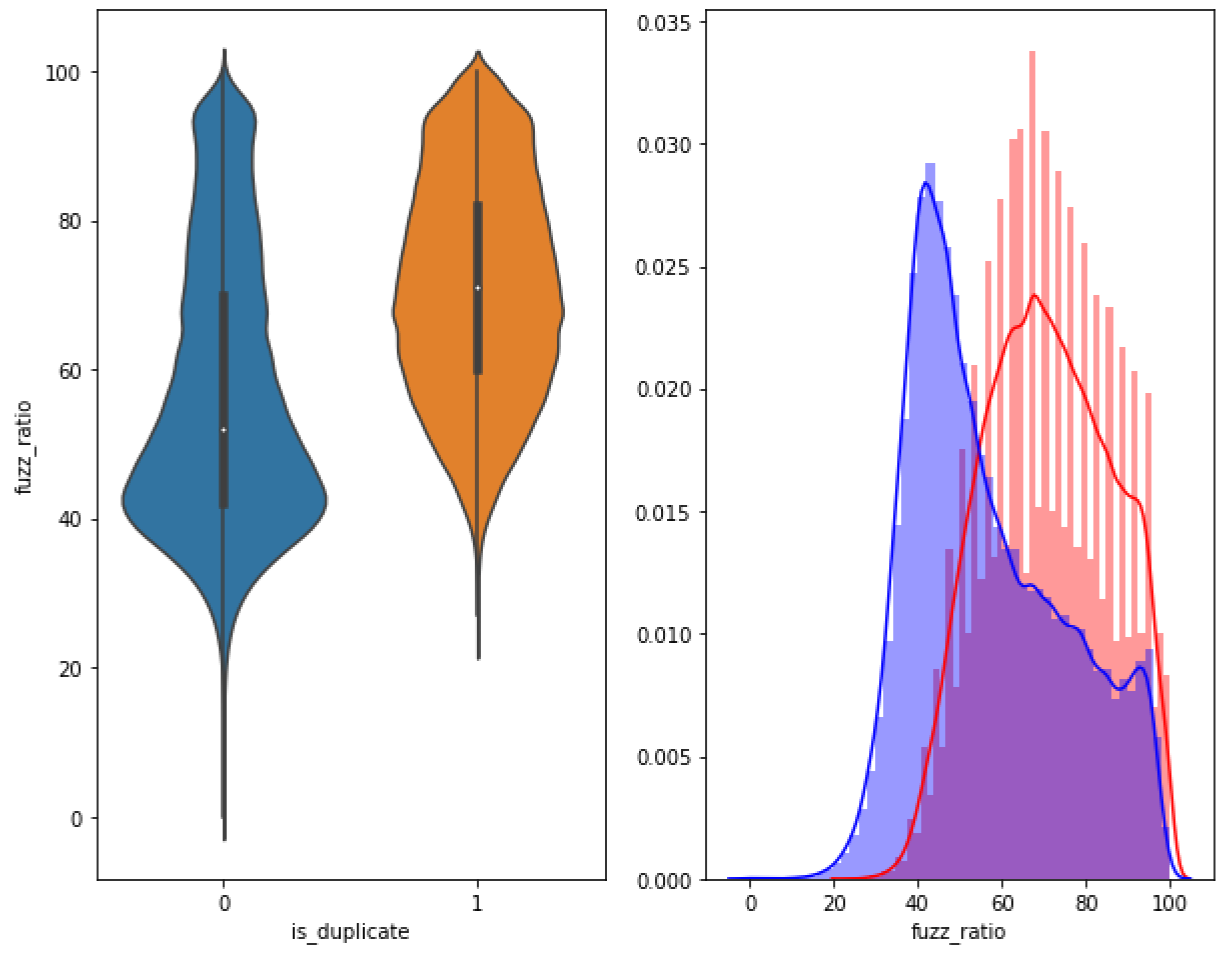
In [0]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```




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



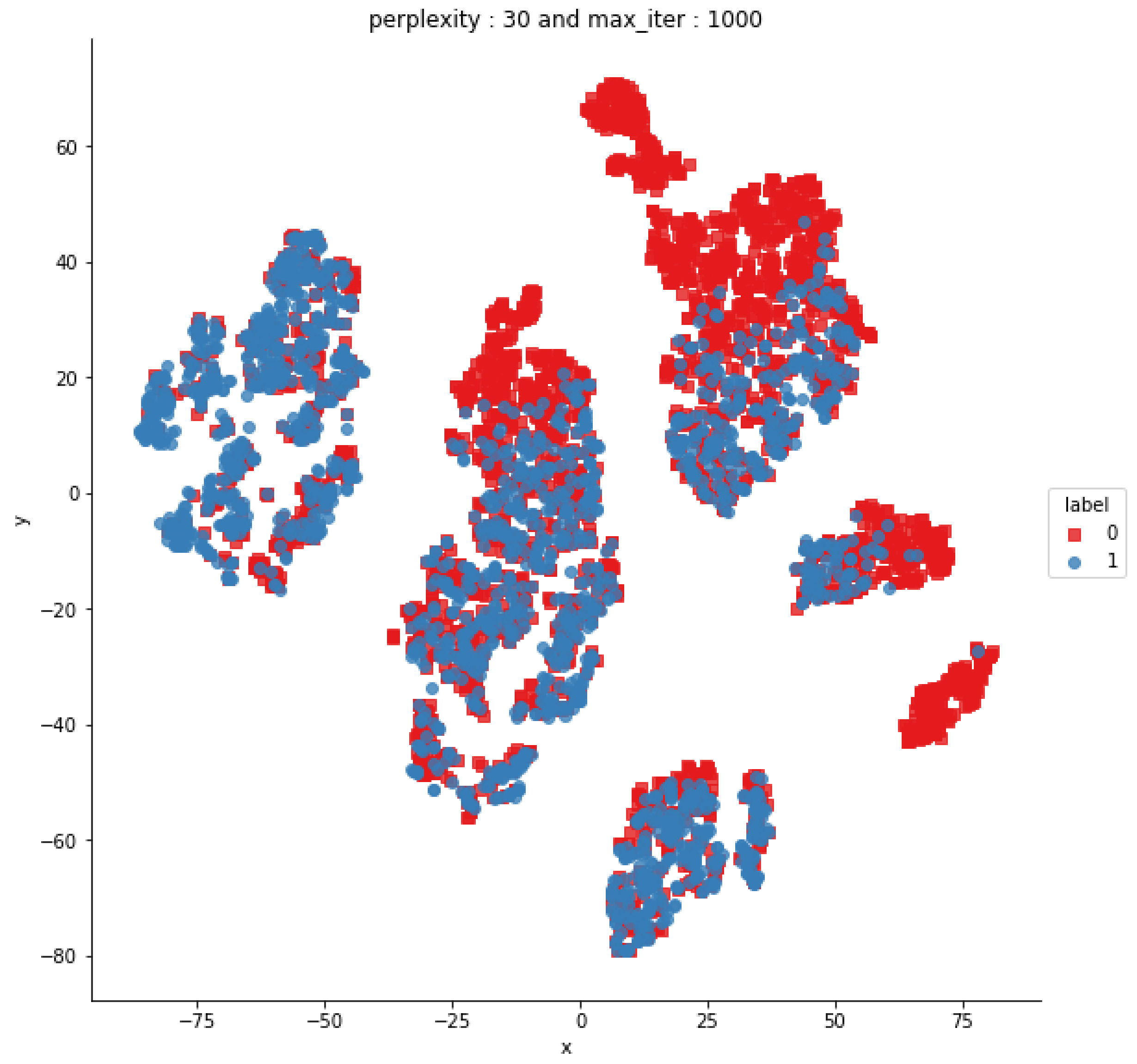

### Visualization

In [0]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_word_eq' , 'abs_len_diff' , 'mean_len' ,
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 11.755s)
[t-SNE] Iteration 1000: error = 0.9432716, gradient norm = 0.0000662 (50 iterations 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=['s','o'])
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')
```

# Text Featurization I

## TFIDF

In [0]:
```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('Quora_question_pair_similarity/data/nlp_features_train.csv'):
    dfnlp = pd.read_csv("Quora_question_pair_similarity/data/nlp_features_train.csv",encoding='latin-1')

if os.path.isfile('Quora_question_pair_similarity/data/df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("Quora_question_pair_similarity/data/df_fe_without_preprocessing_train.csv",encoding='latin-1')
```

In [0]:
```
dfppro.head()
```

Out[7]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common | word_Total | word_share | freq_q1+q2 | freq_q1-q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 14 | 12 | 10.0 | 23.0 | 0.434783 | 2 | 0 |
| **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 | 8 | 13 | 4.0 | 20.0 | 0.200000 | 5 | 3 |
| **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 | 14 | 10 | 4.0 | 24.0 | 0.166667 | 2 | 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 | 1 | 1 | 50 | 65 | 11 | 9 | 0.0 | 19.0 | 0.000000 | 2 | 0 |
| **4** | 4 | 9 | 10 | Which one dissolve in water quikly sugar, salt... | Which fish would survive in salt water? | 0 | 3 | 1 | 76 | 39 | 13 | 7 | 2.0 | 20.0 | 0.100000 | 4 | 2 |

In [0]:
```
dfnlp.head()
```

Out[8]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ... | ctc_max | last_word_eq | first_word_eq | abs_len_diff | mean_len | token_set_ratio | token_sort_ratio | fuzz_ratio | fuzz_partial_ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | what is the step by step guide to invest in sh... | what is the step by step guide to invest in sh... | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | ... | 0.785709 | 0.0 | 1.0 | 2.0 | 13.0 | 100 | 93 | 93 | 10 |
| **1** | 1 | 3 | 4 | what is the story of kohinoor koh i noor dia... | what would happen if the indian government sto... | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | ... | 0.466664 | 0.0 | 1.0 | 5.0 | 12.5 | 86 | 63 | 66 | 7 |
| **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 | 0.249997 | ... | 0.285712 | 0.0 | 1.0 | 4.0 | 12.0 | 66 | 66 | 54 | 5 |
| **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 | 0.000000 | ... | 0.000000 | 0.0 | 0.0 | 2.0 | 12.0 | 36 | 36 | 35 | 4 |
| **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 | 0.666644 | ... | 0.307690 | 0.0 | 1.0 | 6.0 | 10.0 | 67 | 47 | 46 | 5 |

5 rows × 21 columns

In [0]:
```
print('The shape of dfppro is',dfppro.shape)
print('THe shape of dfnlp is',dfnlp.shape)
```

```
The shape of dfppro is (404290, 17)
THe shape of dfnlp is (404290, 21)
```

In [0]:
```
df_feats_nlp = dfnlp.drop(['qid1','qid2'],axis=1)
df_feats_pro = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [0]:
```
df  = df_feats_nlp.merge(df_feats_pro, on='id',how='left')
print('Shape of final merged data',df.shape)
```

```
Shape of final merged data (404290, 30)
```

In [0]:
```
df.head()
```

Out[14]:

| | id | question1 | question2 | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ctc_min | ctc_max | ... | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common | word_Total | word_share | freq_q1+q2 | freq_q1-q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | what is the step by step guide to invest in sh... | what is the step by step guide to invest in sh... | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | 0.916659 | 0.785709 | ... | 1 | 66 | 57 | 14 | 12 | 10.0 | 23.0 | 0.434783 | 2 | 0 |
| **1** | 1 | what is the story of kohinoor koh i noor dia... | what would happen if the indian government sto... | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | 0.699993 | 0.466664 | ... | 1 | 51 | 88 | 8 | 13 | 4.0 | 20.0 | 0.200000 | 5 | 3 |
| **2** | 2 | how can i increase the speed of my internet co... | how can internet speed be increased by hacking... | 0 | 0.399992 | 0.333328 | 0.399992 | 0.249997 | 0.399996 | 0.285712 | ... | 1 | 73 | 59 | 14 | 10 | 4.0 | 24.0 | 0.166667 | 2 | 0 |
| **3** | 3 | why am i mentally very lonely how can i solve... | find the remainder when math 23 24 math i... | 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 1 | 50 | 65 | 11 | 9 | 0.0 | 19.0 | 0.000000 | 2 | 0 |
| **4** | 4 | which one dissolve in water quikly sugar salt... | which fish would survive in salt water | 0 | 0.399992 | 0.199998 | 0.999950 | 0.666644 | 0.571420 | 0.307690 | ... | 1 | 76 | 39 | 13 | 7 | 2.0 | 20.0 | 0.100000 | 4 | 2 |

5 rows × 30 columns

In [0]:
```
df['is_duplicate'].value_counts()
```

Out[13]:
```
0    255027
1    149263
Name: is_duplicate, dtype: int64
```

Observations: We can see that 255027 question pairs that are duplicate, 149263 question pairs that are not duplicate.

In [0]:
```
y_feats = df['is_duplicate']
df.drop(['id','is_duplicate'],axis=1,inplace=True)

x_train,x_test,y_train,y_test = train_test_split(df, y_feats, stratify=y_feats, test_size=0.3)

print('The shape of x_train is {}'.format(x_train.shape))
print('The shape of y_train is {}'.format(y_train.shape))
print('The shape of x_test is {}'.format(x_test.shape))
print('The shape of y_test is {}'.format(y_test.shape))
```

```
The shape of x_train is (283003, 28)
The shape of y_train is (283003,)
The shape of x_test is (121287, 28)
The shape of y_test is (121287,)
```

In [0]:
```
y_train.value_counts()
```

Out[15]:
```
0    178519
1    104484
Name: is_duplicate, dtype: int64
```

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

tfidf_q1 = TfidfVectorizer() #TFIDF vectorizer for question 1
train_q1 = tfidf_q1.fit_transform(x_train['question1'].values.astype('U'))
test_q1 = tfidf_q1.transform(x_test['question1'].values.astype('U'))

tfidf_q2 = TfidfVectorizer() #TFIDF Vectorizer for question 2
train_q2 = tfidf_q2.fit_transform(x_train['question2'].values.astype('U'))
test_q2 = tfidf_q2.transform(x_test['question2'].values.astype('U'))
```

In [0]:
```
train_feats = hstack((train_q1, train_q2)) #Stack train tfidf vectors for question 1 and question 2
test_feats = hstack((test_q1, test_q2)) #Stack test tfidf vectors for question 1 and question 2

print('The shape of TFIDF vectorization of train data is',train_feats.shape)
print('The shape of TFIDF vectorization of test data is',test_feats.shape)
```

```
The shape of TFIDF vectorization of train data is (283003, 111664)
The shape of TFIDF vectorization of test data is (121287, 111664)
```

In [0]:
```
x_train.drop(['question1','question2'], axis=1, inplace=True) #Drop unwanted columns
x_test.drop(['question1','question2'], axis=1, inplace=True)
```

In [0]:
```
X_train=x_train.as_matrix() #Convert to matrix
X_test=x_test.as_matrix()
```

In [0]:
```
X_train = hstack((X_train,train_feats)) #Finally feature engineering with tfidf features of question 1 and 2
X_test = hstack((X_test,test_feats))
```

# Machine Learning Models I

## Random Model

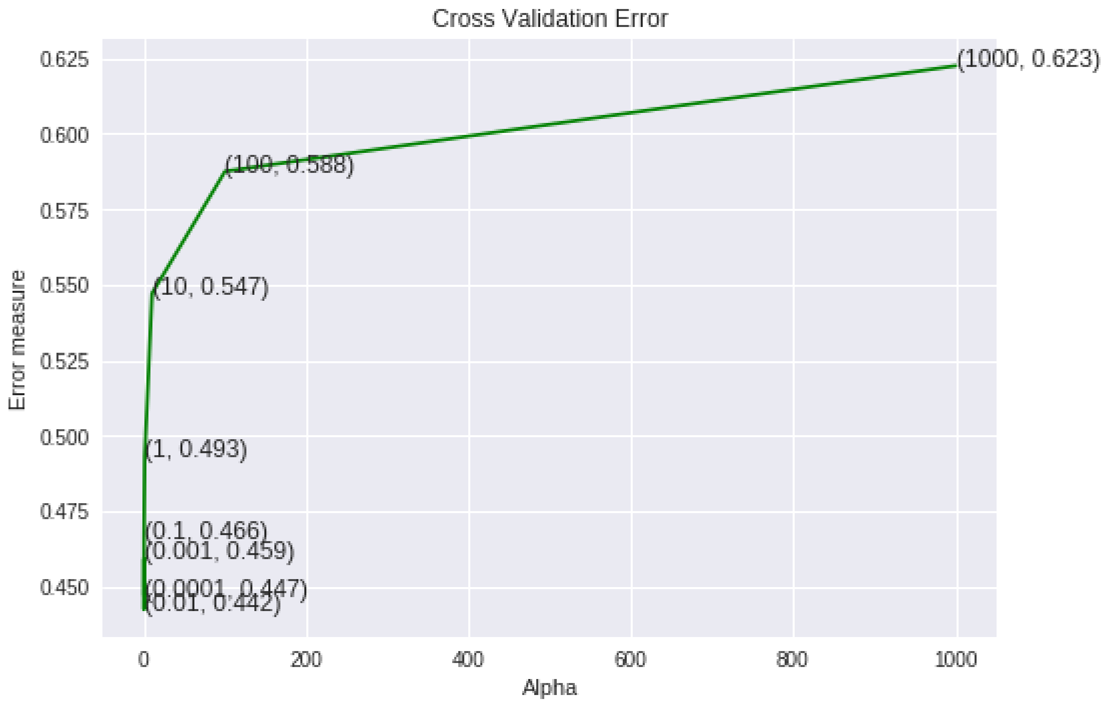
In [0]:
```
predicted_y = np.zeros((len(y_test),2))
for i in range(len(y_test)):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Random model Log Loss on Test data is ",log_loss(y_test, predicted_y, eps=1e-15))

predicted_y =np.argmax(predicted_y, axis=1)
```

```
Random model Log Loss on Test data is  0.884447493006593
```

## Logistic Regression

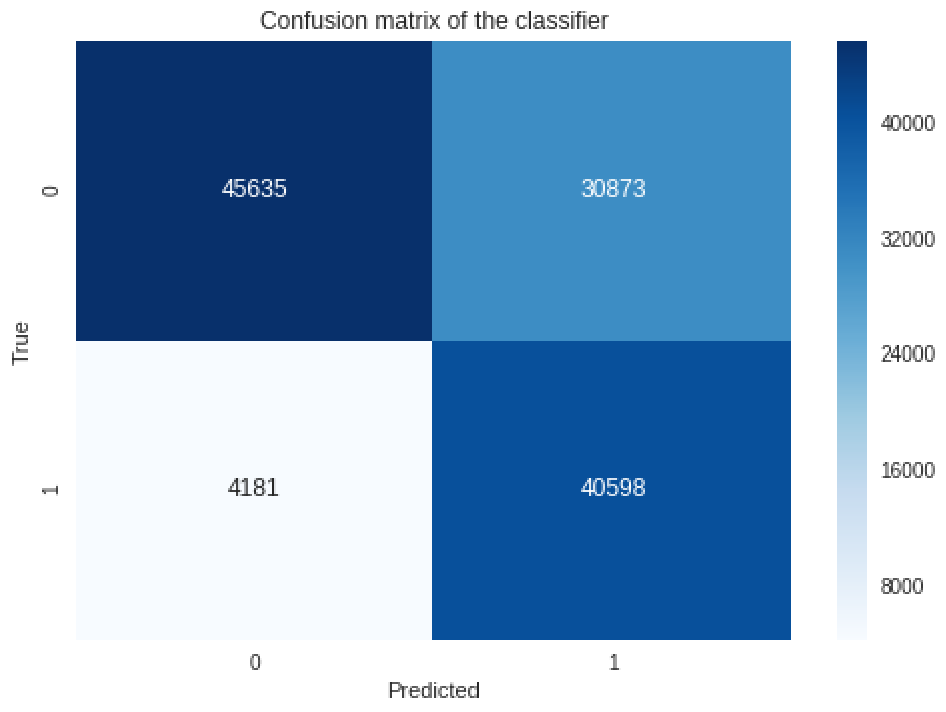
In [0]:
```
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]

predict_train = []
predict_test = []
log_error_test_array=[]
log_error_train_array=[]

for i in alpha:
    clf = SGDClassifier(alpha=i, loss='log')
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y_test = sig_clf.predict_proba(X_test)
    predict_y_train = sig_clf.predict_proba(X_train)
    predict_test.append(clf.predict(X_test))
    predict_train.append(clf.predict(X_train))
    log_error_test_array.append(log_loss(y_test, predict_y_test, labels=clf.classes_, eps=1e-15))
    log_error_train_array.append(log_loss(y_train, predict_y_train, labels=clf.classes_, eps=1e-15))
    print('When alpha is {} log loss is {}'.format(i,log_loss(y_test, predict_y_test, labels=clf.classes_, eps=1e-15)))

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


best_alpha = np.argmin(log_error_test_array)

print('For values of best alpha {} the train log loss is {}'.format(alpha[best_alpha],log_error_train_array[best_alpha]))

print('For values of best alpha {} the test log loss is {}'.format(alpha[best_alpha],log_error_test_array[best_alpha]))
```

```
When alpha is 0.0001 log loss is 0.44702936689993683
When alpha is 0.001 log loss is 0.4592529411350289
When alpha is 0.01 log loss is 0.44236647075139723
When alpha is 0.1 log loss is 0.4662027092416626
When alpha is 1 log loss is 0.4932871094479345
When alpha is 10 log loss is 0.547060351646075
When alpha is 100 log loss is 0.5875748231327537
When alpha is 1000 log loss is 0.6225942575558097
```




```
For values of best alpha 0.01 the train log loss is 0.4409423224089968
For values of best alpha 0.01 the test log loss is 0.44236647075139723
```

In [0]:
```
from pandas_ml import ConfusionMatrix
y_true = np.array(y_test)
y_pred = predict_test[best_alpha]
#print(confusion_matrix(y_test, y_pred))
cm = ConfusionMatrix(y_test,y_pred) #This the confusion matrix of pandas_ml which provides interesting s
confusion_matrix_plot = confusion_matrix(y_test,y_pred) #We are plotting confusion matrix of sklearn
heatmap = sns.heatmap(confusion_matrix_plot, annot=True,cmap='Blues', fmt='g')
plt.title('Confusion matrix of the classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("*"*50)
print("The True Positive Rate observed is",cm.TPR) #This prints the True Positive Rate of the confusion matrix (using pandas_ml confusion matrix).
print("The True Negative Rate observed is",cm.TNR)
print("The False Positive Rate observed is",cm.FPR)
print("The False Negative Rate observed is",cm.FNR)
print("*"*50)
print("The stats observed for confusion matrix are:")
cm.print_stats()#Prints all the stats of the confusion matrix plotted (using pandas_ml confusion matrix)
```

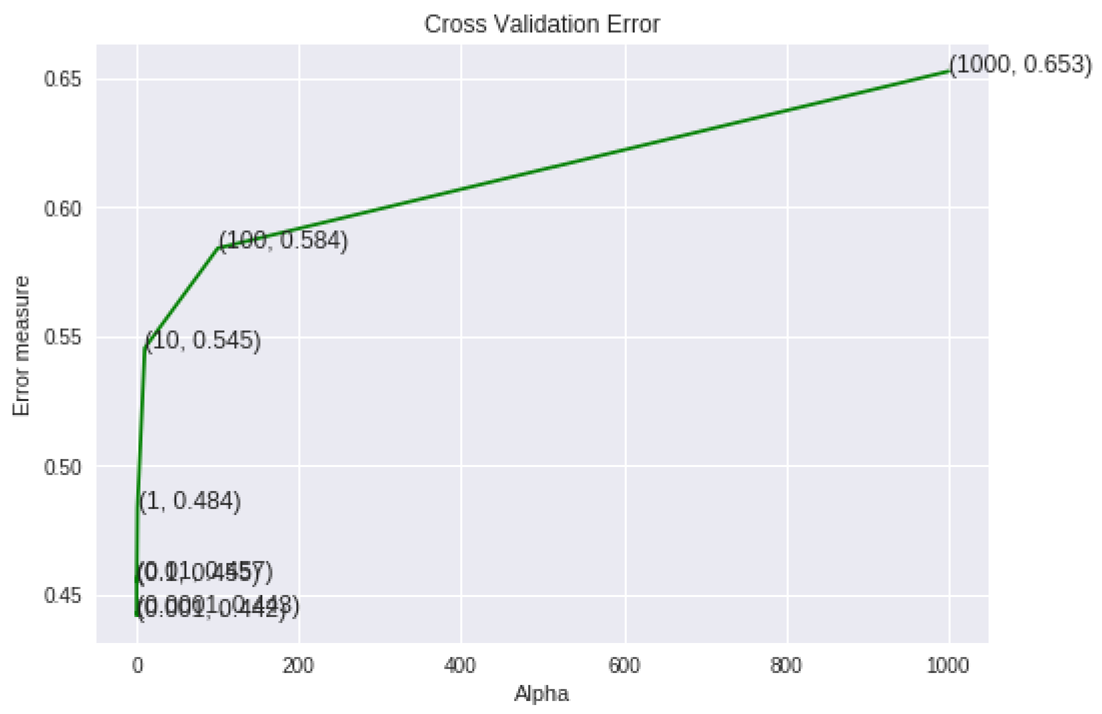



```
**************************************************
The True Positive Rate observed is 0.5926146515783204
The True Negative Rate observed is 0.4156324267229997
The False Positive Rate observed is 0.5843675732770003
The False Negative Rate observed is 0.40738534842167956
**************************************************
The stats observed for confusion matrix are:
population: 36154
P: 13432
N: 22722
PositiveTest: 21238
NegativeTest: 14916
TP: 7960
TN: 9444
FP: 13278
FN: 5472
TPR: 0.5926146515783204
TNR: 0.4156324267229997
PPV: 0.37479988699500894
NPV: 0.6331456154465004
FPR: 0.5843675732770003
FDR: 0.625200113004991
FNR: 0.40738534842167956
ACC: 0.4813851855949549
F1_score: 0.4591866166714739
MCC: 0.008094886088047079
informedness: 0.008247078301319988
markedness: 0.007945502441509378
prevalence: 0.37152182331139016
LRP: 1.0141128267180746
LRN: 0.9801577601479673
DOR: 1.0346424503796015
FOR: 0.36685438455349966
```

## Linear SVM

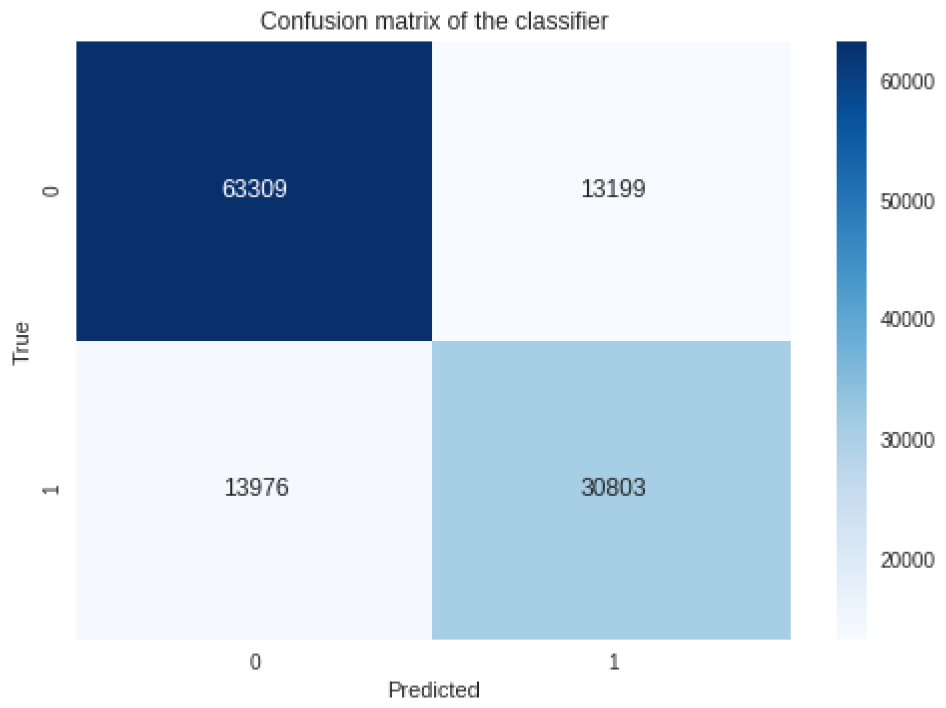
In [0]:
```
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]

predict_train = []
predict_test = []
log_error_test_array=[]
log_error_train_array=[]

for i in alpha:
    clf = SGDClassifier(alpha=i, loss='hinge')
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y_test = sig_clf.predict_proba(X_test)
    predict_y_train = sig_clf.predict_proba(X_train)
    predict_test.append(clf.predict(X_test))
    predict_train.append(clf.predict(X_train))
    log_error_test_array.append(log_loss(y_test, predict_y_test, labels=clf.classes_, eps=1e-15))
    log_error_train_array.append(log_loss(y_train, predict_y_train, labels=clf.classes_, eps=1e-15))
    print('When alpha is {} log loss is {}'.format(i,log_loss(y_test, predict_y_test, labels=clf.classes_, eps=1e-15)))

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


best_alpha = np.argmin(log_error_test_array)

print('For values of best alpha {} the train log loss is {}'.format(alpha[best_alpha],log_error_train_array[best_alpha]))

print('For values of best alpha {} the test log loss is {}'.format(alpha[best_alpha],log_error_test_array[best_alpha]))
```

```
When alpha is 0.0001 log loss is 0.44304936605114953
When alpha is 0.001 log loss is 0.44207942163087655
When alpha is 0.01 log loss is 0.4566526366972352
When alpha is 0.1 log loss is 0.4553373139363503
When alpha is 1 log loss is 0.48381342523053994
When alpha is 10 log loss is 0.5454418921782209
When alpha is 100 log loss is 0.5841136291131219
When alpha is 1000 log loss is 0.6526084011103247
```




```
For values of best alpha 0.001 the train log loss is 0.442749096808023
For values of best alpha 0.001 the test log loss is 0.44207942163087655
```

In [0]:
```
from pandas_ml import ConfusionMatrix
y_true = np.array(y_test)
y_pred = predict_test[best_alpha]
#print(confusion_matrix(y_test, y_pred))
cm = ConfusionMatrix(y_test,y_pred) #This the confusion matrix of pandas_ml which provides interesting s
confusion_matrix_plot = confusion_matrix(y_test,y_pred) #We are plotting confusion matrix of sklearn
heatmap = sns.heatmap(confusion_matrix_plot, annot=True,cmap='Blues', fmt='g')
plt.title('Confusion matrix of the classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("*"*50)
print("The True Positive Rate observed is",cm.TPR) #This prints the True Positive Rate of the confusion matrix (using pandas_ml confusion matrix).
print("The True Negative Rate observed is",cm.TNR)
print("The False Positive Rate observed is",cm.FPR)
print("The False Negative Rate observed is",cm.FNR)
print("*"*50)
print("The stats observed for confusion matrix are:")
cm.print_stats()#Prints all the stats of the confusion matrix plotted (using pandas_ml confusion matrix)
```




```
**************************************************
The True Positive Rate observed is 0.3631870158613058
The True Negative Rate observed is 0.6399841577187115
The False Positive Rate observed is 0.3600158422812885
The False Negative Rate observed is 0.6368129841386942
**************************************************
The stats observed for confusion matrix are:
population: 36279
P: 13555
N: 22724
PositiveTest: 13104
NegativeTest: 23175
TP: 4923
TN: 14543
FP: 8181
FN: 8632
TPR: 0.3631870158613058
TNR: 0.6399841577187115
PPV: 0.3756868131868132
NPV: 0.627529665587918
FPR: 0.3600158422812885
FDR: 0.6243131868131868
FNR: 0.6368129841386942
ACC: 0.5365638523663827
F1_score: 0.36933118271503057
MCC: 0.0031937458432245293
informedness: 0.0031711735800172836
markedness: 0.0032164787747310797
prevalence: 0.37363212877973484
LRP: 1.0088084278734033
LRN: 0.9950449186252965
DOR: 1.0138320481723797
FOR: 0.372470334412082
```

# Text Featurization II

## TFIDF Word2Vec

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[3]:

| | 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
# 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.
- here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usage/vectors-similarity (https://spacy.io/usage/vectors-similarity)
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [0]:
```python
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 384])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)
```

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

In [0]:
```python
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 384])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df['q2_feats_m'] = list(vecs2)
```

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

In [0]:
```python
#prepro_features_train.csv (Simple Preprocessing Feartures)
#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')

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
```

In [0]:
```python
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]:
```python
# dataframe of nlp features
df1.head()
```

Out[9]:

| | id | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ctc_min | ctc_max | last_word_eq | first_word_eq | abs_len_diff | mean_len | token_set_ratio | token_sort_ratio | fuzz_ratio | fuzz_partial_ratio | longest_substr_ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | 0.916659 | 0.785709 | 0.0 | 1.0 | 2.0 | 13.0 | 100 | 93 | 93 | 100 | 0.982759 |
| **1** | 1 | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | 0.699993 | 0.466664 | 0.0 | 1.0 | 5.0 | 12.5 | 86 | 63 | 66 | 75 | 0.596154 |
| **2** | 2 | 0 | 0.399992 | 0.333328 | 0.399992 | 0.249997 | 0.399996 | 0.285712 | 0.0 | 1.0 | 4.0 | 12.0 | 66 | 66 | 54 | 54 | 0.166667 |
| **3** | 3 | 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.0 | 2.0 | 12.0 | 36 | 36 | 35 | 40 | 0.039216 |
| **4** | 4 | 0 | 0.399992 | 0.199998 | 0.999950 | 0.666644 | 0.571420 | 0.307690 | 0.0 | 1.0 | 6.0 | 10.0 | 67 | 47 | 46 | 56 | 0.175000 |

In [0]:
```python
# data before preprocessing
df2.head()
```

Out[10]:

| | id | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common | word_Total | word_share | freq_q1+q2 | freq_q1-q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 1 | 66 | 57 | 14 | 12 | 10.0 | 23.0 | 0.434783 | 2 | 0 |
| **1** | 1 | 4 | 1 | 51 | 88 | 8 | 13 | 4.0 | 20.0 | 0.200000 | 5 | 3 |
| **2** | 2 | 1 | 1 | 73 | 59 | 14 | 10 | 4.0 | 24.0 | 0.166667 | 2 | 0 |
| **3** | 3 | 1 | 1 | 50 | 65 | 11 | 9 | 0.0 | 19.0 | 0.000000 | 2 | 0 |
| **4** | 4 | 3 | 1 | 76 | 39 | 13 | 7 | 2.0 | 20.0 | 0.100000 | 4 | 2 |

In [0]:
```python
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[11]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 374 | 375 | 376 | 377 | 378 | 379 | 380 | 381 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 121.929927 | 100.083900 | 72.497894 | 115.641800 | -48.370870 | 34.619058 | -172.057787 | -92.502617 | 113.223315 | 50.562441 | ... | 12.397642 | 40.909519 | 8.150261 | -15.170692 | 18.007709 | 6.166999 | -30.124163 | 3.700902 | -1 |
| **1** | -78.070939 | 54.843781 | 82.738482 | 98.191872 | -51.234859 | 55.013510 | -39.140730 | -82.692352 | 45.161489 | -9.556289 | ... | -21.987077 | -12.389279 | 20.667979 | 2.202714 | -17.142454 | -5.880972 | -10.123963 | -4.890663 | -13 |
| **2** | -5.355015 | 73.671810 | 14.376365 | 104.130241 | 1.433537 | 35.229116 | -148.519385 | -97.124595 | 41.972195 | 50.948731 | ... | 3.027700 | 14.025767 | -2.960312 | -3.206544 | 4.355141 | 2.936152 | -20.199555 | 9.816351 | 11 |
| **3** | 5.778359 | -34.712038 | 48.999631 | 59.699204 | 40.661263 | -41.658731 | -36.808594 | 24.170655 | 0.235600 | -29.407290 | ... | 13.100007 | 1.405670 | -1.891076 | -7.882638 | 18.000561 | 12.106918 | -10.507835 | 5.243834 | 10 |
| **4** | 51.138220 | 38.587312 | 123.639488 | 53.333041 | -47.062739 | 37.356212 | -298.722753 | -106.421119 | 106.248914 | 65.880707 | ... | 13.906532 | 43.461721 | 11.519207 | -22.468284 | 45.431128 | 8.161224 | -35.373910 | 7.728865 | 9 |

5 rows × 384 columns

In [0]:
```python
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[12]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 374 | 375 | 376 | 377 | 378 | 379 | 380 | 381 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 125.983301 | 95.636485 | 42.114702 | 95.449980 | -37.386295 | 39.400078 | -148.116070 | -87.851475 | 110.371966 | 62.272814 | ... | 16.165592 | 33.030668 | 7.019996 | -14.793959 | 15.437511 | 8.199658 | -25.070834 | 1.571619 | 1.603 |
| **1** | -106.871904 | 80.290331 | 79.066297 | 59.302092 | -42.175328 | 117.616655 | -144.364237 | -127.131513 | 22.962533 | 25.397575 | ... | -4.901128 | -4.565393 | 41.520751 | -0.727564 | -16.413776 | -7.373778 | 2.638877 | -7.403457 | 2.703 |
| **2** | 7.072875 | 15.513378 | 1.846914 | 85.937583 | -33.808811 | 94.702337 | -122.256856 | -114.009530 | 53.922293 | 60.131814 | ... | 8.359966 | -2.165985 | 10.936580 | -16.531660 | 14.681230 | 15.633759 | -1.210901 | 14.183826 | 11.703 |
| **3** | 39.421531 | 44.136989 | -24.010929 | 85.265863 | -0.339022 | -9.323137 | -60.499651 | -37.044763 | 49.407848 | -23.350150 | ... | 3.311411 | 3.788879 | 13.398598 | -6.592596 | 6.437365 | 5.993293 | 2.732392 | -3.727647 | 5.614 |
| **4** | 31.950101 | 62.854106 | 1.778164 | 36.218768 | -45.130875 | 66.674880 | -106.342341 | -22.901008 | 59.835938 | 62.663961 | ... | -2.403870 | 11.991204 | 8.088483 | -15.090201 | 8.375166 | 1.727225 | -6.601129 | 11.317413 | 11.544 |

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]+df3_q2.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]:
```
final_features = pd.read_csv('final_features.csv')
final_features.shape
```

Out[6]:
```
(404290, 797)
```

In [0]:
```
y_feats = final_features['is_duplicate']
final_features.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace=True)
```

In [0]:
```
y_feats.shape
```

Out[8]:
```
(404290,)
```

In [0]:
```
final_features.head()
```

Out[8]:

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.999980 | 0.833319 | 0.999983 | 0.999983 | 0.916659 | 0.785709 | 0.0 | 1.0 | 2.0 | 13.0 | ... | 16.165592 | 33.030668 | 7.019996 | -14.793959 | 15.437511 | 8.199658 | -25.070834 | 1.571619 | 1.603738 |
| **1** | 0.799984 | 0.399996 | 0.749981 | 0.599988 | 0.699993 | 0.466664 | 0.0 | 1.0 | 5.0 | 12.5 | ... | -4.901128 | -4.565393 | 41.520751 | -0.727564 | -16.413776 | -7.373778 | 2.638877 | -7.403457 | 2.703070 |
| **2** | 0.399992 | 0.333328 | 0.399992 | 0.249997 | 0.399996 | 0.285712 | 0.0 | 1.0 | 4.0 | 12.0 | ... | 8.359966 | -2.165985 | 10.936580 | -16.531660 | 14.681230 | 15.633759 | -1.210901 | 14.183826 | 11.703135 |
| **3** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.0 | 2.0 | 12.0 | ... | 3.311411 | 3.788879 | 13.398598 | -6.592596 | 6.437365 | 5.993293 | 2.732392 | -3.727647 | 5.614115 |
| **4** | 0.399992 | 0.199998 | 0.999950 | 0.666644 | 0.571420 | 0.307690 | 0.0 | 1.0 | 6.0 | 10.0 | ... | -2.403870 | 11.991204 | 8.088483 | -15.090201 | 8.375166 | 1.727225 | -6.601129 | 11.317413 | 11.544603 |

5 rows × 794 columns

In [0]:
```
final_features['is_duplicate'] = y_feats.tolist()
p = final_features.groupby('is_duplicate')
final_features = pd.concat([p.get_group(0).sample(75000),p.get_group(1).sample(75000)])
y_feats = final_features['is_duplicate']
#final_features.drop(columns = ['is_duplicate'],inplace=True)
final_features.drop(['is_duplicate'], axis=1, inplace=True)
```

In [0]:
```
x_train,x_test, y_train, y_test = train_test_split(final_features, y_feats, stratify=y_feats, test_size=0.3)
```

In [0]:
```
y_train.value_counts()
```

Out[11]:
```
1    52500
0    52500
Name: is_duplicate, dtype: int64
```

In [0]:
```
print("Number of data points in train data :",x_train.shape)
print("Number of data points in test data :",x_test.shape)
```

```
Number of data points in train data : (105000, 794)
Number of data points in test data : (45000, 794)
```

In [0]:
```
print("Number of data points in train data :",x_train.shape)
print("Number of data points in test data :",x_test.shape)
```

```
Number of data points in train data : (105000, 794)
Number of data points in test data : (45000, 794)
```

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[1])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.5 Class 1:  0.5
---------- Distribution of output variable in train data ----------
Class 0:  0.5 Class 1:  0.5
```

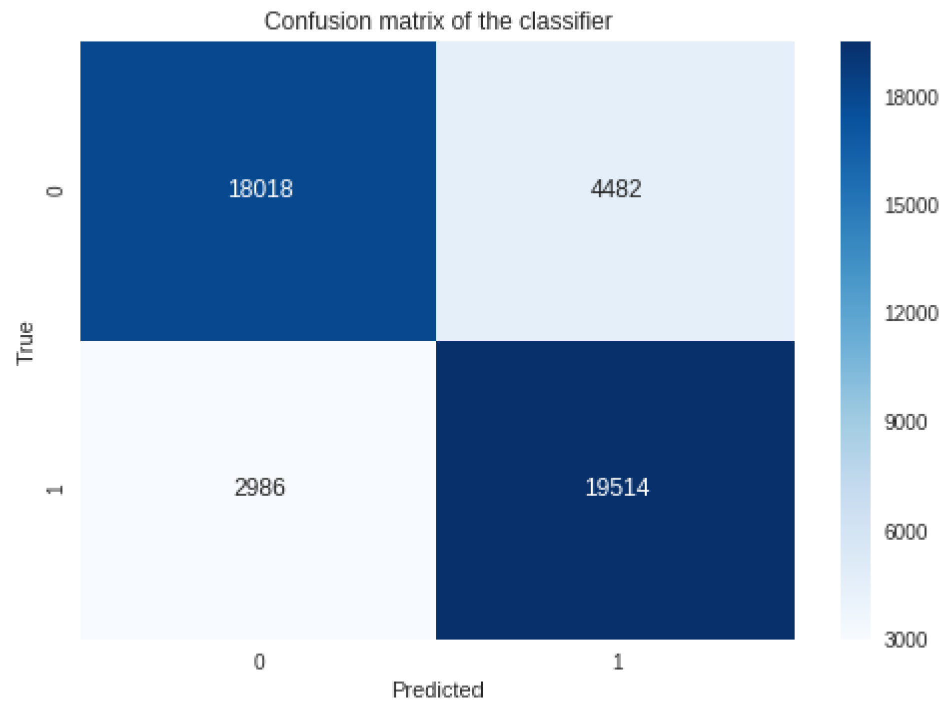
# Machine Learning Models II

## XGBoost

In [0]:
```
import xgboost as xgb
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import ShuffleSplit, RandomizedSearchCV
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,log_loss,hamming_loss,classification_report
import joblib

#my_cv = ShuffleSplit(n_splits = 5).split(x_train)
param_grid = {'n_estimators' : list(range(100,301,100)), 'max_depth' : list(range(1,6,2)), 'learning_rate' : [0.01, 0.1]} #Parameters for grid search
model = xgb.XGBClassifier()
rsearch_log = RandomizedSearchCV(estimator = model, param_distributions = param_grid, scoring = 'neg_log_loss', cv=3)
rsearch_log.fit(x_train,y_train)

print("The optimal base learners for XGBoost using RandomizedSearchCV is",rsearch_log.best_params_['n_estimators'])
print("The optimal max depth for XGBoost using RandomizedSearchCV is",rsearch_log.best_params_['max_depth'])
print("The optimal learning rate for XGBoost using RandomizedSearchCV is",rsearch_log.best_params_['learning_rate'])
print('*'*50)

rpred = rsearch_log.predict(x_test)
rpred_train = rsearch_log.predict(x_train)

sig_clf = CalibratedClassifierCV(rsearch_log, method="sigmoid") #To get probability of each class
sig_clf.fit(x_train, y_train)

predict_y_train = sig_clf.predict_proba(x_train)
predict_y_test = sig_clf.predict_proba(x_test)

print('\nThe test log loss of XGBoost is {}'.format(log_loss(y_test,predict_y_test)))
print('\nThe train log loss of XGBoost is {}\n'.format(log_loss(y_train,predict_y_train)))


print('\nThe test accuracy of XGBoost is {}%'.format(accuracy_score(y_test, rpred) * 100))
print('\nThe train accuracy of XGBoost is {}%\n'.format(accuracy_score(y_train, rpred_train) * 100))
print('\nThe test precision of XGBoost is {}%'.format(precision_score(y_test,rpred)*100))
print('\nThe train precision of XGBoost is {}%\n'.format(precision_score(y_train,rpred_train)*100))
print('\nThe test recall of XGBoost is {}%'.format(recall_score(y_test,rpred)*100))
print('\nThe train recall of XGBoost is {}%\n'.format(recall_score(y_train,rpred_train)*100))
print('\nThe test f1 score of XGBoost is {}%'.format(f1_score(y_test,rpred)*100))
print('\nThe train f1_score of XGBoost is {}%\n'.format(f1_score(y_train,rpred_train)*100))
print('\nThe test hamming of XGBoost is {}'.format(hamming_loss(y_test,rpred)))
print('\nThe train hamming of XGBoost is {}\n'.format(hamming_loss(y_train,rpred_train)))
print('*'*50)
print(classification_report(y_test,rpred))
print("*"*50)
print(classification_report(y_train,rpred_train))
```

```
The optimal base learners for XGBoost using RandomizedSearchCV is 300
The optimal max depth for XGBoost using RandomizedSearchCV is 5
The optimal learning rate for XGBoost using RandomizedSearchCV is 0.1
**************************************************

The test log loss of XGBoost is 0.349438718329845

The train log loss of XGBoost is 0.2601238790231136


The test accuracy of XGBoost is 83.40444444444445%

The train accuracy of XGBoost is 89.32476190476191%


The test precision of XGBoost is 81.32188698116353%

The train precision of XGBoost is 87.09016761583098%


The test recall of XGBoost is 86.72888888888889%

The train recall of XGBoost is 92.33714285714287%


The test f1 score of XGBoost is 83.93840330350997%

The train f1_score of XGBoost is 89.6369368453168%


The test hamming of XGBoost is 0.16595555555555555

The train hamming of XGBoost is 0.10675238095238095

**************************************************
              precision    recall  f1-score   support

           0       0.86      0.80      0.83     22500
           1       0.81      0.87      0.84     22500

   micro avg       0.83      0.83      0.83     45000
   macro avg       0.84      0.83      0.83     45000
weighted avg       0.84      0.83      0.83     45000

**************************************************
              precision    recall  f1-score   support

           0       0.92      0.86      0.89     52500
           1       0.87      0.92      0.90     52500

   micro avg       0.89      0.89      0.89    105000
   macro avg       0.89      0.89      0.89    105000
weighted avg       0.89      0.89      0.89    105000
```

In [0]:
```
from pandas_ml import ConfusionMatrix
y_true = np.array(y_test)
y_pred = np.array(rpred)
#print(confusion_matrix(y_test, y_pred))
cm = ConfusionMatrix(y_test,y_pred) #This the confusion matrix of pandas_ml which provides interesting s
confusion_matrix_plot = confusion_matrix(y_test,y_pred) #We are plotting confusion matrix of sklearn
heatmap = sns.heatmap(confusion_matrix_plot, annot=True,cmap='Blues', fmt='g')
plt.title('Confusion matrix of the classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("*"*50)
print("The True Positive Rate observed is",cm.TPR) #This prints the True Positive Rate of the confusion matrix (using pandas_ml confusion matrix).
print("The True Negative Rate observed is",cm.TNR)
print("The False Positive Rate observed is",cm.FPR)
print("The False Negative Rate observed is",cm.FNR)
print("*"*50)
print("The stats observed for confusion matrix are:")
cm.print_stats()#Prints all the stats of the confusion matrix plotted (using pandas_ml confusion matrix)
```




```
**************************************************
The True Positive Rate observed is 0.5351157222665602
The True Negative Rate observed is 0.47107438016528924
The False Positive Rate observed is 0.5289256198347108
The False Negative Rate observed is 0.46488427773343977
**************************************************
The stats observed for confusion matrix are:
population: 5047
P: 2506
N: 2541
PositiveTest: 2685
NegativeTest: 2362
TP: 1341
TN: 1197
FP: 1344
FN: 1165
TPR: 0.5351157222665602
TNR: 0.47107438016528924
PPV: 0.4994413407821229
NPV: 0.5067739204064352
FPR: 0.5289256198347108
FDR: 0.5005586592178771
FNR: 0.46488427773343977
ACC: 0.5028729938577373
F1_score: 0.5166634559815064
MCC: 0.006202669054356645
informedness: 0.006190102431849365
markedness: 0.0062152611885579745
prevalence: 0.49653259361997226
LRP: 1.0117031624102155
LRN: 0.9868596071183546
DOR: 1.0251743562231759
FOR: 0.4932260795935648
```

## Summary and Results

1) Exploratory data analysis is performed and null values are filled up.

2) After getting a sense of data we perform basic feature extraction such as word length, common words, frequency of words etc..

3) Later advanced feature extraction such as fuzz ratio, token ratio etc..

4) After feature engineering we perform text featurization like TFIDF and apply Logistic Regression and Linear SVM on it.

5) The performance seems okay but it can be improved.

6) Another text featurization technique TFIDF weighted Word2vec is performed and XGBoost is applied on it.

7) XGBoost obtained the best performance so far among all models.

In [0]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ['Model', 'Vectorization', 'Hyperparameter', 'Train Log Loss', 'Test Log Loss']
x.add_row(['Logistic Regression', 'TFIDF', 'alpha = 0.01', 0.4409, 0.4423])
x.add_row(['','','','',''])
x.add_row(['Linear SVM', 'TFIDF', 'alpha = 0.001', 0.4427, 0.4420])
x.add_row(['','','','',''])
x.add_row(['XGBOOST', 'TFIDF weighted word2vec', 'n_estimators = 300', 0.2601, 0.3494])
x.add_row(['','','max_depth = 5','',''])
x.add_row(['','','learning_rate = 0.1','',''])
print(x.get_string())
```

```
+---------------------+-------------------------+---------------------+----------------+---------------+
|        Model        |      Vectorization      |    Hyperparameter   | Train Log Loss | Test Log Loss |
+---------------------+-------------------------+---------------------+----------------+---------------+
| Logistic Regression |          TFIDF          |     alpha = 0.01    |     0.4409     |     0.4423    |
|                     |                         |                     |                |               |
|      Linear SVM     |          TFIDF          |    alpha = 0.001    |     0.4427     |     0.442     |
|                     |                         |                     |                |               |
|       XGBOOST       | TFIDF weighted word2vec |  n_estimators = 300 |     0.2601     |     0.3494    |
|                     |                         |    max_depth = 5    |                |               |
|                     |                         | learning_rate = 0.1 |                |               |
+---------------------+-------------------------+---------------------+----------------+---------------+
```