Quora Question Similarity2

December 20, 2018

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

0.0.1 Problem Statement

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

1.2 Sources/Useful Links

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

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.

3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data
2.1.1 Data Overview

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

2.1.2 Example Data point
2.2 Mapping the real world problem to an ML problem
2.2.1 Type of Machine Leaning Problem
It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.
2.2.2 Performance Metric
Source: https://www.kaggle.com/c/quora-question-pairs#evaluation
Metric(s): * log-loss : https://www.kaggle.com/wiki/LogarithmicLoss * Binary Confusion Matrix
2.3 Train and Test Construction
We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

3. Exploratory Data Analysis

```
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[0]:    id  qid1  qid2                                          question1  \
        0   0     1     2  What is the step by step guide to invest in sh...
        1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...
        2   2     5     6  How can I increase the speed of my internet co...
        3   3     7     8  Why am I mentally very lonely? How can I solve...
        4   4     9    10  Which one dissolve in water quikly sugar, salt...

                                                  question2  is_duplicate
        0  What is the step by step guide to invest in sh...             0
        1  What would happen if the Indian government sto...             0
        2  How can Internet speed be increased by hacking...             0
        3  Find the remainder when [math]23^{24}[/math] i...             0
        4            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.

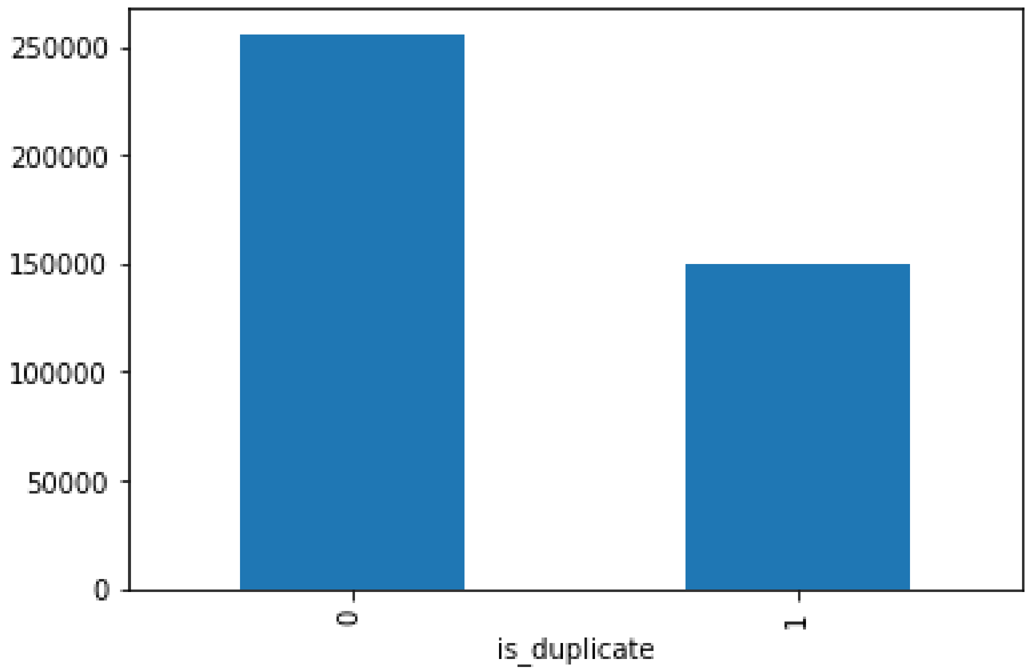
0.1 3.2 Detailed Stats

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[0]: <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 - rou
        print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is
```

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

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

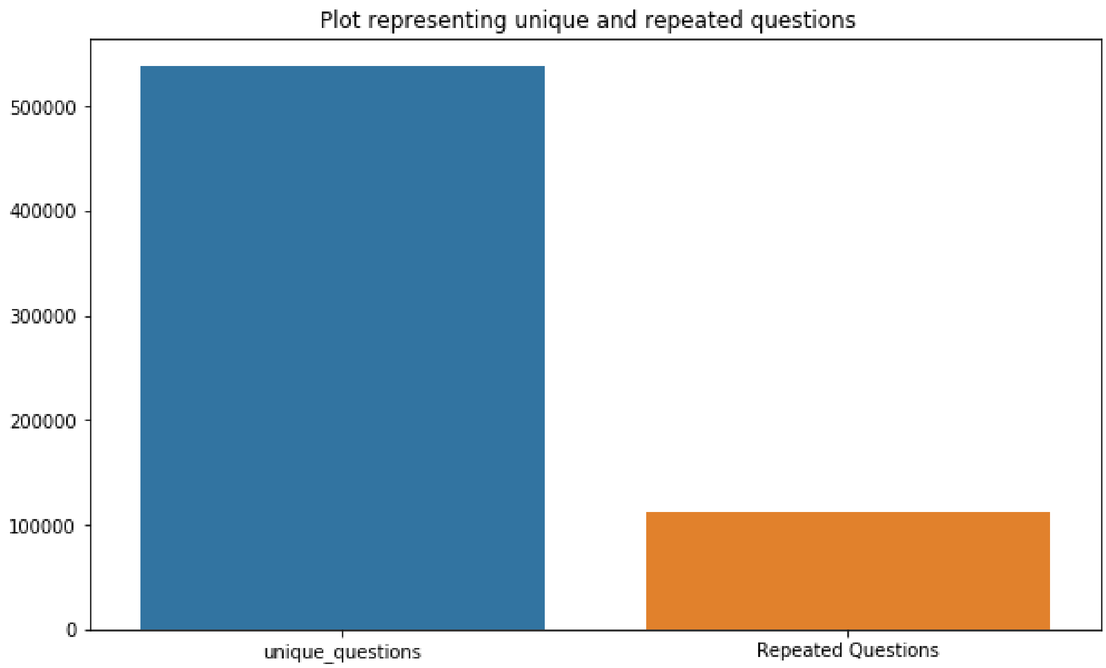
3.2.2 Number of unique questions

```
In [0]: qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
        unique_qs = len(np.unique(qids))
```

```
        qs_morethan_onetime = np.sum(qids.value_counts() > 1)
        print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
        #print len(np.unique(qids))

        print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(q

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

        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

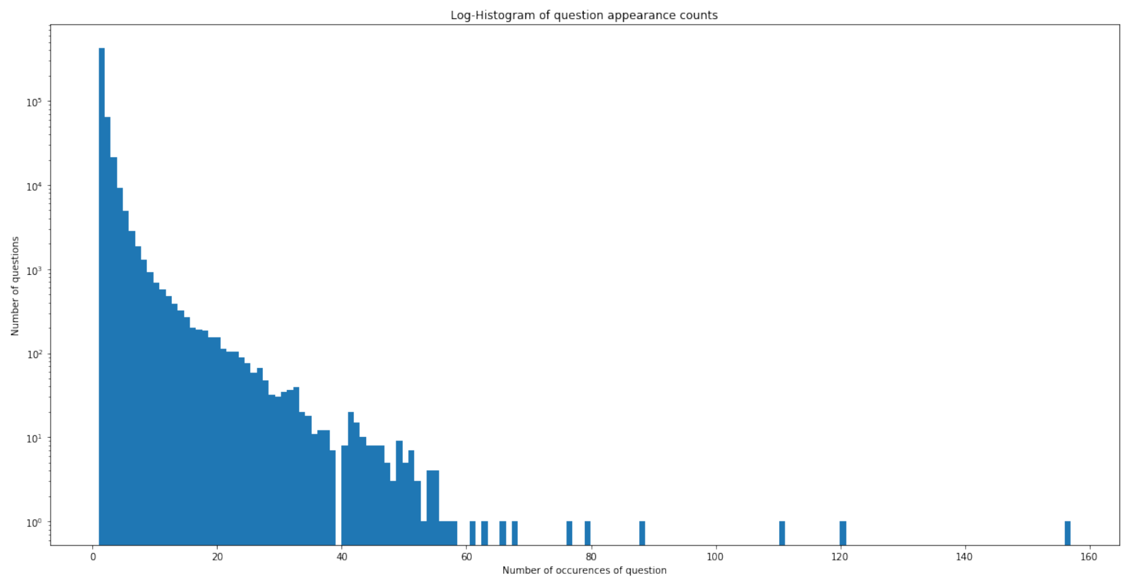
```
In [0]: #checking whether there are any repeated pair of questions

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

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

```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

```
In [0]: plt.figure(figsize=(20, 10))

        plt.hist(qids.value_counts(), bins=160)

        plt.yscale('log', nonposy='clip')

        plt.title('Log-Histogram of question appearance counts')

        plt.xlabel('Number of occurences of question')

        plt.ylabel('Number of questions')

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

```
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)
```

```python
    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[0]:    id  qid1  qid2                                          question1  \
0   0     1     2  What is the step by step guide to invest in sh...
1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...
2   2     5     6  How can I increase the speed of my internet co...
3   3     7     8  Why am I mentally very lonely? How can I solve...
4   4     9    10  Which one dissolve in water quikly sugar, salt...

                                          question2  is_duplicate  freq_qid1  \
0  What is the step by step guide to invest in sh...             0          1
1  What would happen if the Indian government sto...             0          4
2  How can Internet speed be increased by hacking...             0          1
3  Find the remainder when [math]23^{24}[/math] i...             0          1
4            Which fish would survive in salt water?             0          3

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

   word_share  freq_q1+q2  freq_q1-q2
0    0.434783           2           0
1    0.200000           5           3
2    0.166667           2           0
3    0.000000           2           0
4    0.100000           4           2
```

3.3.1 Analysis of some of the extracted features

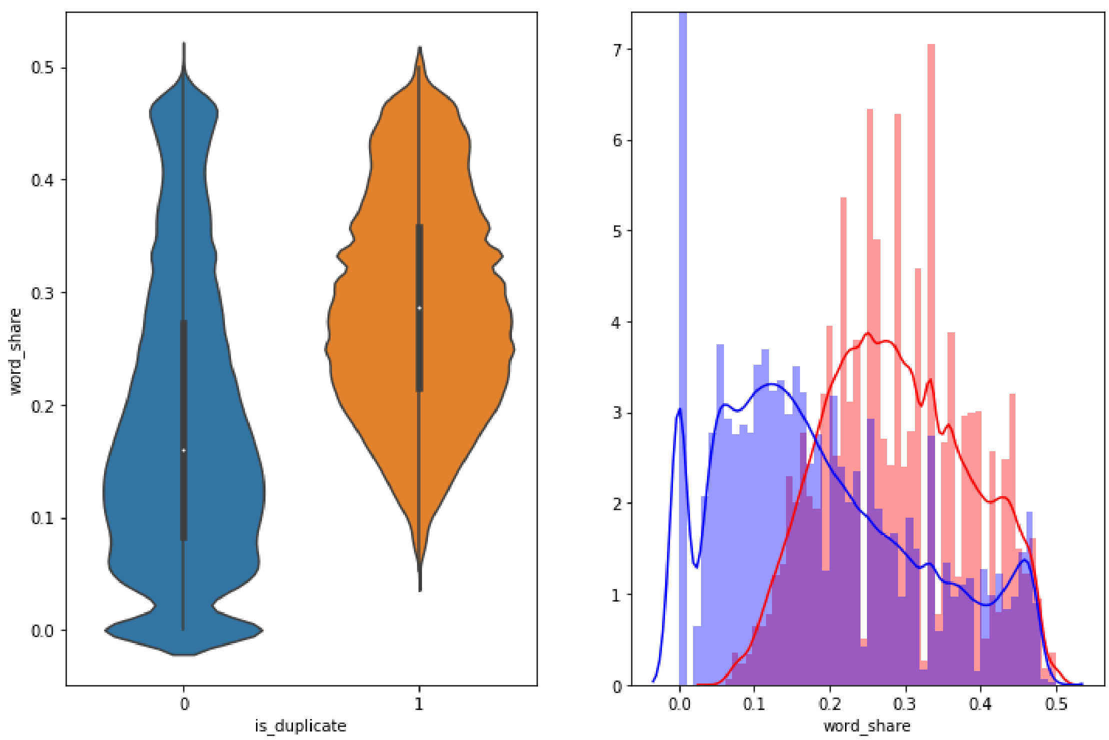
- Here are some questions have only one single words.

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

```
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

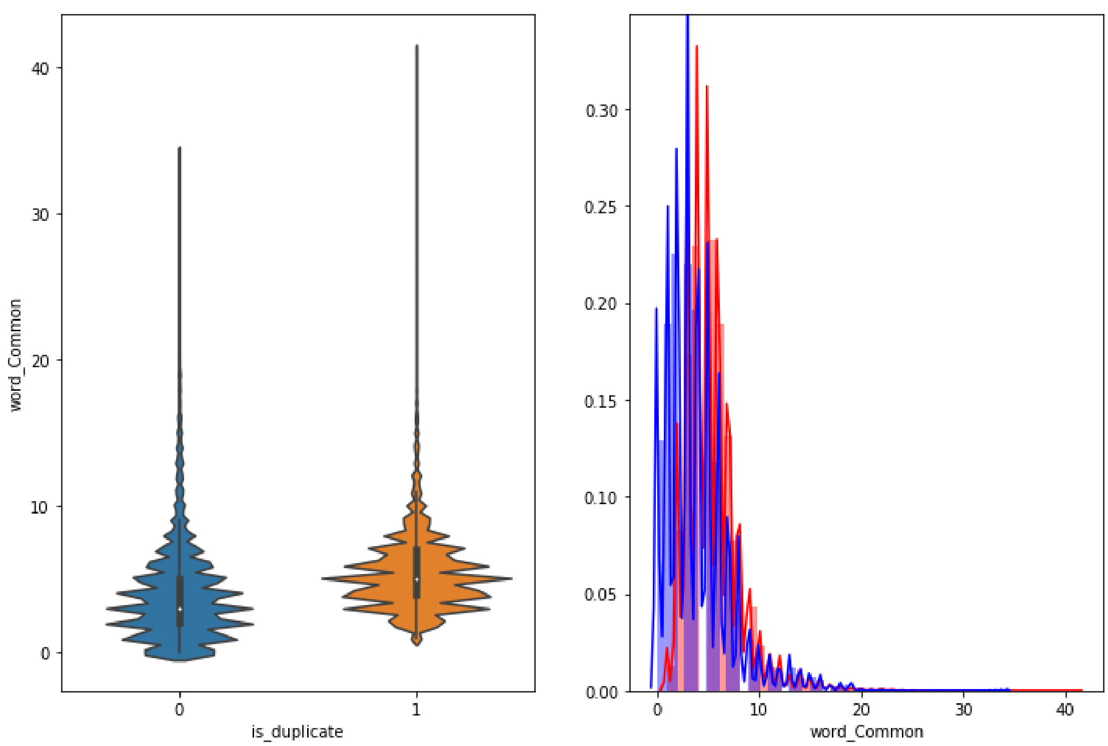
In [0]:
```
plt.figure(figsize=(12, 8))

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



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

0.2 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-decod
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 no
```

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

Out[0]:
```
   id  qid1  qid2                                          question1  \
0   0     1     2  What is the step by step guide to invest in sh...
```

```
1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...

                                         question2  is_duplicate  freq_qid1  \
0  What is the step by step guide to invest in sh...             0          1
1  What would happen if the Indian government sto...             0          4

   freq_qid2  q1len  q2len  q1_n_words  q2_n_words  word_Common  word_Total  \
0          1     66     57          14          12         10.0        23.0
1          1     51     88           8          13          4.0        20.0

   word_share  freq_q1+q2  freq_q1-q2
0    0.434783           2           0
1    0.200000           5           3
```

3.4 Preprocessing of Text

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

In [0]:
```python
# 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("n't", " not").replace("what's", "what is").replac
                           .replace("'ve", " have").replace("i'm", "i am").replace("'r
                           .replace("he's", "he is").replace("she's", "she is").replac
                           .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)
```

```python
        if type(x) == type(''):
            x = porter.stem(x)
            example1 = BeautifulSoup(x)
            x = example1.get_text()


        return x
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

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

Features: - **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2 cwc_min = common_word_count / (min(len(q1_words), len(q2_words)) - **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2 cwc_max = common_word_count / (max(len(q1_words), len(q2_words)) - **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2 csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops)) - **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2csc_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 Q2ctc_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 Q2ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))
- **last_word_eq** : Check if First word of both questions is equal or notlast_word_eq = int(q1_tokens[-1] == q2_tokens[-1])
- **first_word_eq** : Check if First word of both questions is equal or notfirst_word_eq = int(q1_tokens[0] == q2_tokens[0])
- **abs_len_diff** : Abs. length differenceabs_len_diff = abs(len(q1_tokens) - len(q2_tokens))
- **mean_len** : Average Token Length of both Questionsmean_len = (len(q1_tokens) + len(q2_tokens))/2
- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage 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 Q2longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [0]:
```
def get_token_features(q1, q2):
    token_features = [0.0]*10

    # Converting the Sentence into Tokens:
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])

    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])

    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))

    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))

    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_I
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_I
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_I
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_I
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAI
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAI

    # 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
```

```python
    return token_features

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

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

    print("token features...")

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"
    # The token sort approach involves tokenizing the string in question, sorting the
    # then joining them back into a string We then compare the transformed strings with
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["que
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"]
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["questi
    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[0]:    id  qid1  qid2                                         question1  \
        0   0     1     2  what is the step by step guide to invest in sh...
        1   1     3     4  what is the story of kohinoor  koh i noor  dia...

                                                   question2  is_duplicate   cwc_min  \
        0  what is the step by step guide to invest in sh...             0  0.999980
        1  what would happen if the indian government sto...             0  0.799984

            cwc_max   csc_min   csc_max          ...             ctc_max  last_word_eq  \
        0  0.833319  0.999983  0.999983          ...            0.785709           0.0
        1  0.399996  0.749981  0.599988          ...            0.466664           0.0

           first_word_eq  abs_len_diff  mean_len  token_set_ratio  token_sort_ratio  \
        0            1.0           2.0      13.0              100                93
        1            1.0           5.0      12.5               86                63

           fuzz_ratio  fuzz_partial_ratio  longest_substr_ratio
        0          93                 100              0.982759
        1          66                  75              0.596154

        [2 rows x 21 columns]
```

3.5.1 Analysis of extracted features
3.5.1.1 Plotting Word clouds

- Creating Word Cloud of Duplicates and Non-Duplicates Question pairs
- We can observe the most frequent occuring words

```
In [0]: df_duplicate = df[df['is_duplicate'] == 1]
        dfp_nonduplicate = df[df['is_duplicate'] == 0]

        # Converting 2d array of q1 and q2 and flatten the array: like {{1,2},{3,4}} to {1,2,3
        p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
        n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).flatten(

        print ("Number of data points in class 1 (duplicate pairs) :",len(p))
        print ("Number of data points in class 0 (non duplicate pairs) :",len(n))

        #Saving the np array into a text file
```

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

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

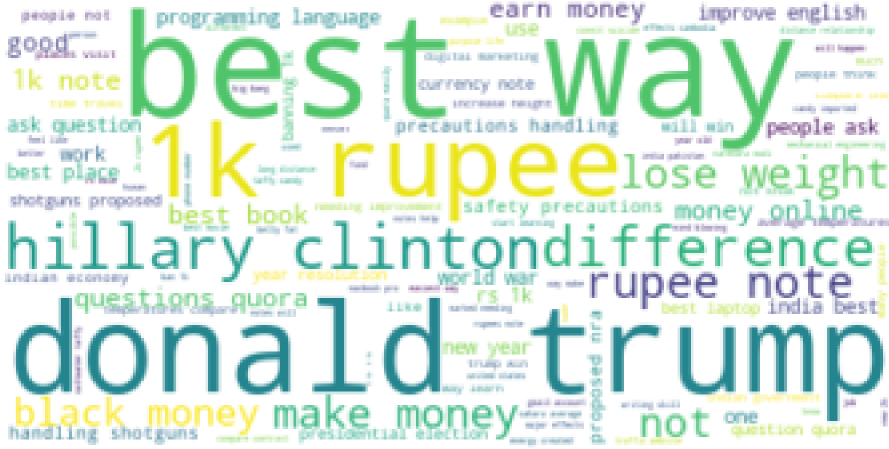
```
In [0]: # reading the text files and removing the Stop Words:
        d = path.dirname('.')

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

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

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

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

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

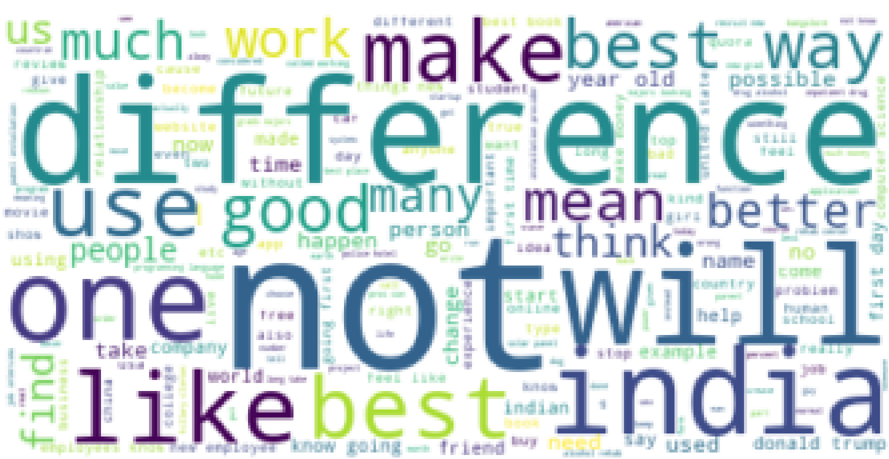
```
Word Cloud for Duplicate Question pairs
```



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

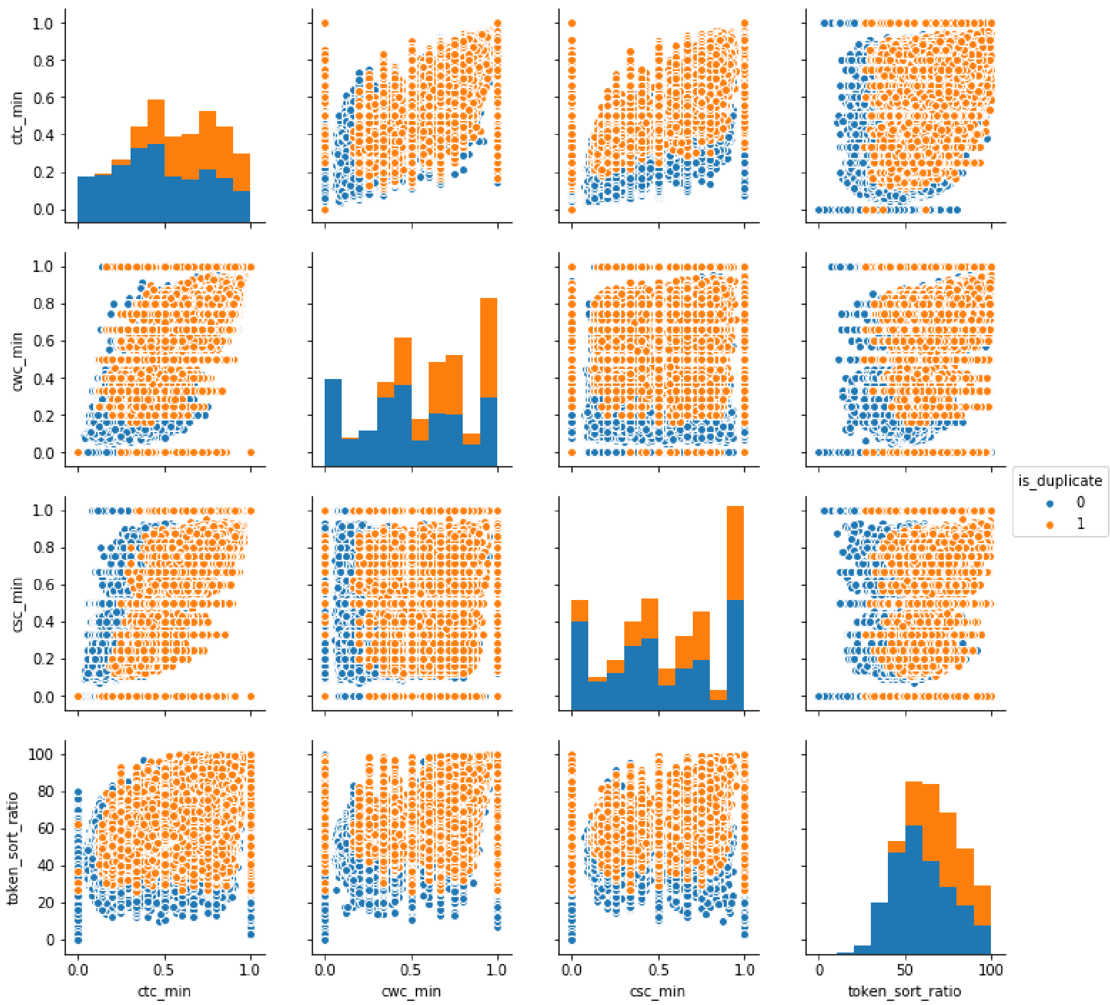
```
In [0]: wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
        # generate word cloud
        wc.generate(textn_w)
        print ("Word Cloud for non-Duplicate Question pairs:")
        plt.imshow(wc, interpolation='bilinear')
        plt.axis("off")
        plt.show()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

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

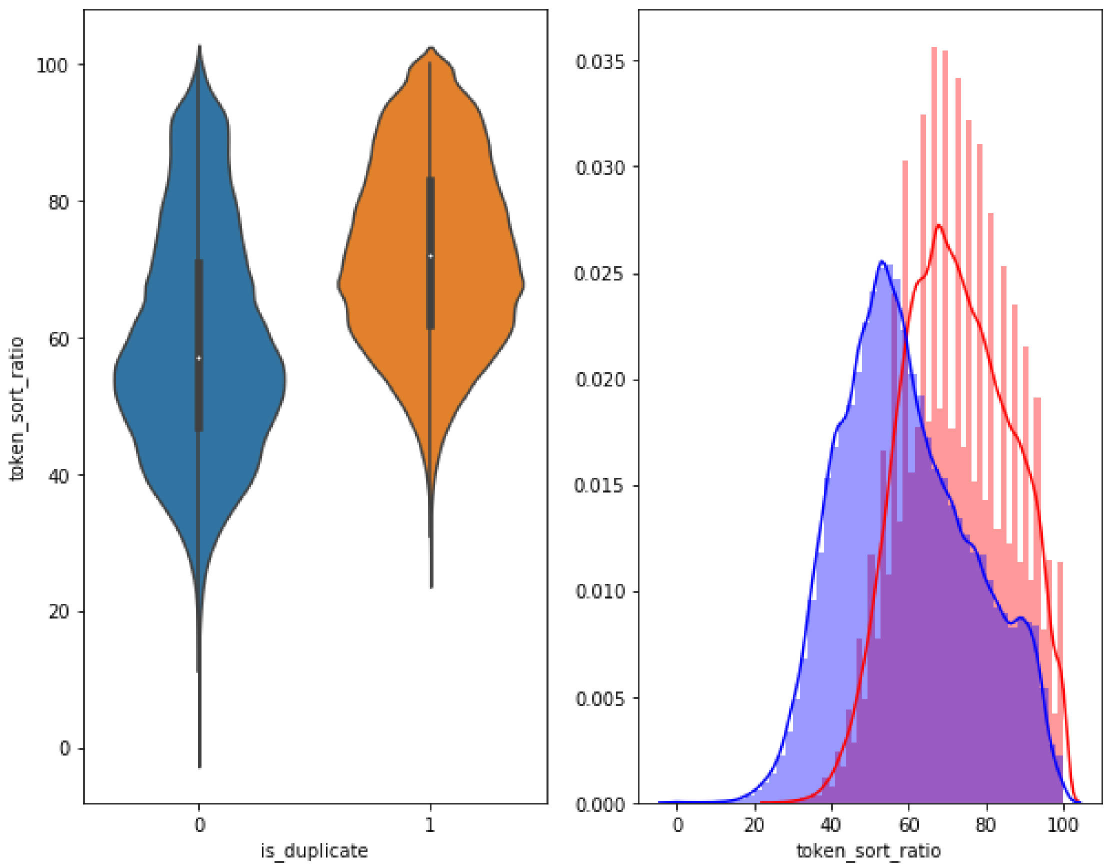


3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

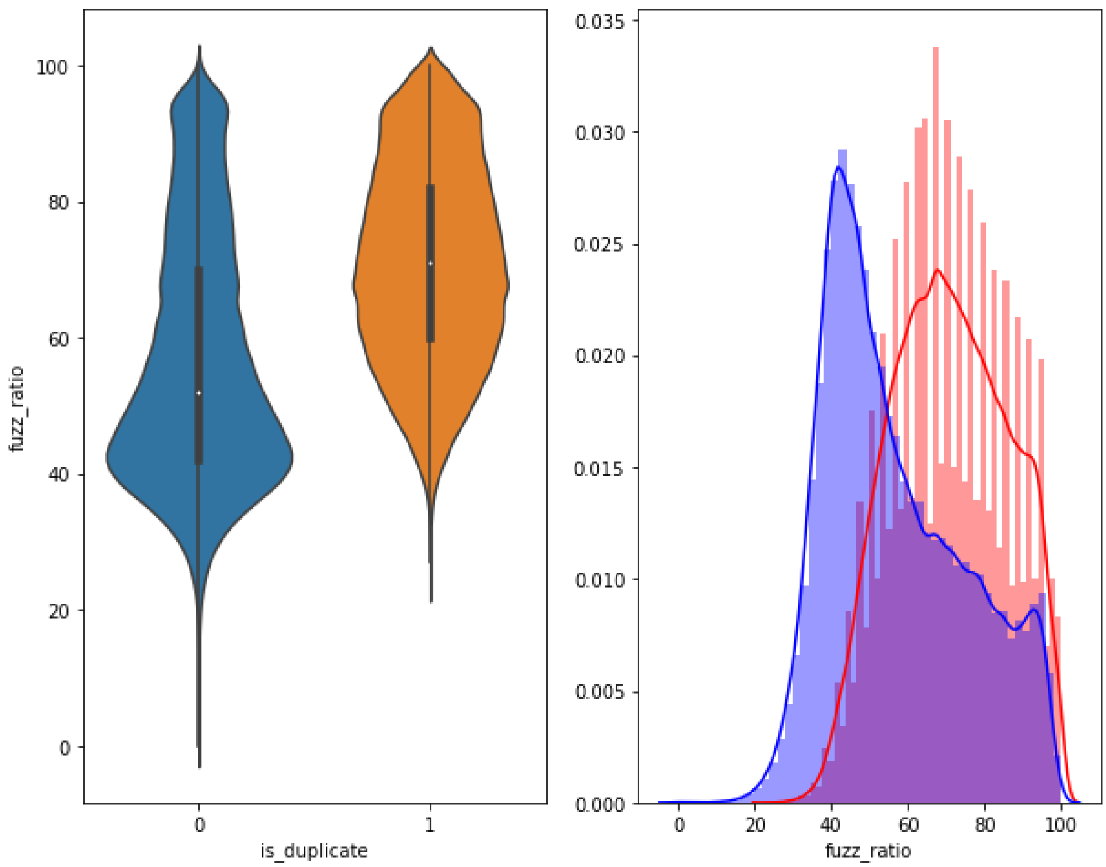
```
In [0]: n = df.shape[0]
        sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']]
        plt.show()
```




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



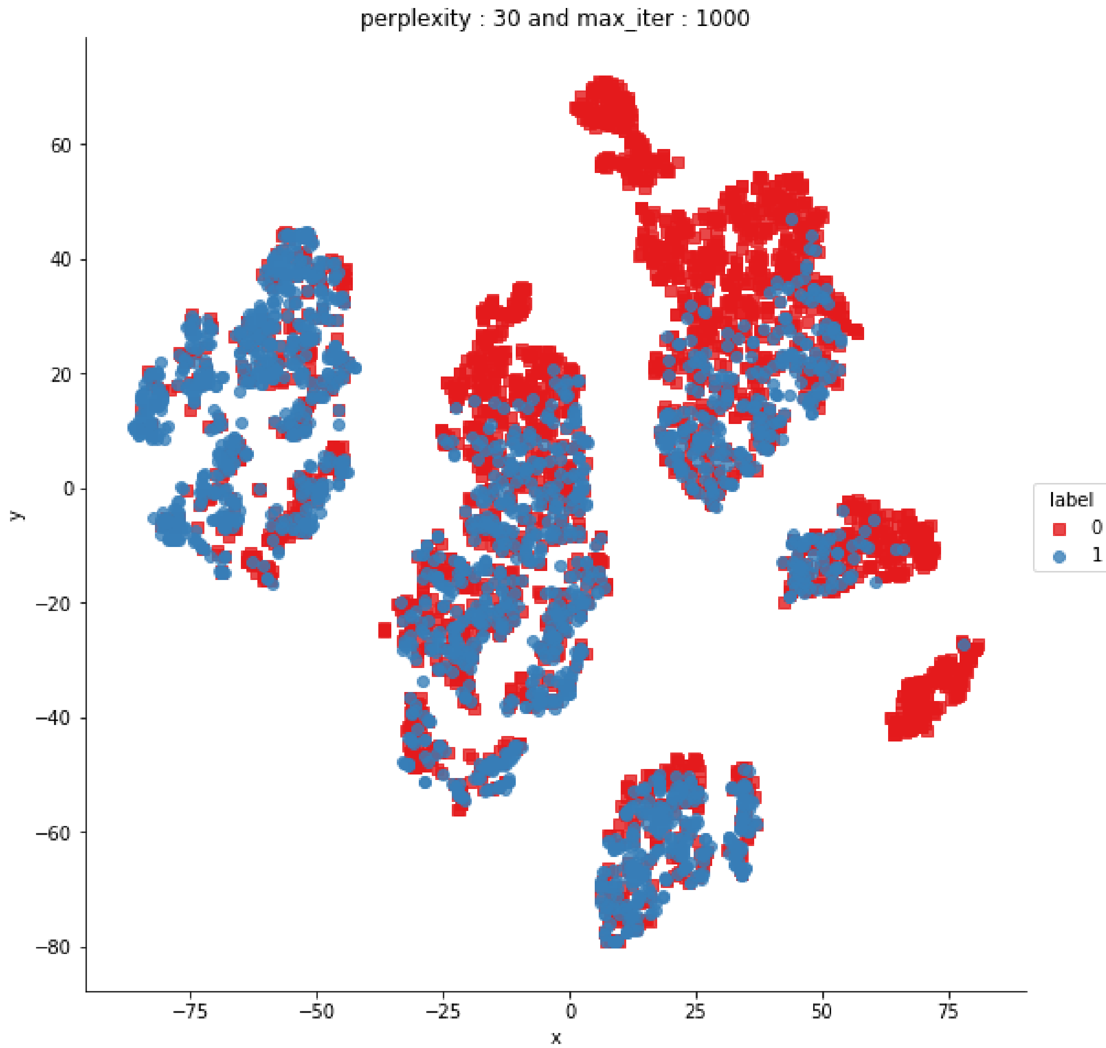
3.5.2 Visualization

```
In [0]: # Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the

        from sklearn.preprocessing import MinMaxScaler

        dfp_subsampled = df[0:5000]
        X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_
        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",ma
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]:
```python
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 TF-IDF vectors

In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
import seaborn as sns
from nltk.corpus import stopwords
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

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

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

In [9]:
```
# 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 [10]:
```
df.head()
```

Out[10]:
```
   id  qid1  qid2                                          question1  \
0   0     1     2  What is the step by step guide to invest in sh...
1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...
2   2     5     6  How can I increase the speed of my internet co...
3   3     7     8  Why am I mentally very lonely? How can I solve...
4   4     9    10  Which one dissolve in water quikly sugar, salt...

                                           question2  is_duplicate
0  What is the step by step guide to invest in sh...             0
1  What would happen if the Indian government sto...             0
2  How can Internet speed be increased by hacking...             0
3  Find the remainder when [math]23^{24}[/math] i...             0
4            Which fish would survive in salt water?             0
```

In [11]:
```
y_true=df['is_duplicate']
```

In [12]:
```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
# TF-IDF Vectorizing
tfidf1 = TfidfVectorizer(lowercase=False,ngram_range=(1,1))
tfidfq1 = tfidf1.fit_transform(list(df['question1']))
tfidf2 = TfidfVectorizer(lowercase=False,ngram_range=(1,1))
tfidfq2 = tfidf2.fit_transform(list(df['question2']))

print "Tfidf features of Q1: ",tfidfq1.get_shape()
print "Tfidf features of Q2: ",tfidfq2.get_shape()
```

```
Tfidf features of Q1:  (404290, 84717)
Tfidf features of Q2:  (404290, 78351)
CPU times: user 12.5 s, sys: 260 ms, total: 12.7 s
Wall time: 12.9 s
```

In [13]:
```
from scipy.sparse import hstack
g=hstack([tfidfq1,tfidfq2])
print "Q1 & Q2 features combined together: ",g.shape
```

```
Q1 & Q2 features combined together:  (404290, 163068)
```

In [14]:
```
print type(g)
```

```
<class 'scipy.sparse.coo.coo_matrix'>
```

In [15]:
```
#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')
```

```
    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 n
```

In [16]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [17]:
```
# dataframe of nlp features
df1.head()
```

Out[17]:
```
   id  is_duplicate   cwc_min   cwc_max   csc_min   csc_max   ctc_min  \
0   0             0  0.999980  0.833319  0.999983  0.999983  0.916659
1   1             0  0.799984  0.399996  0.749981  0.599988  0.699993
2   2             0  0.399992  0.333328  0.399992  0.249997  0.399996
3   3             0  0.000000  0.000000  0.000000  0.000000  0.000000
4   4             0  0.399992  0.199998  0.999950  0.666644  0.571420

    ctc_max  last_word_eq  first_word_eq  abs_len_diff  mean_len  \
0  0.785709           0.0            1.0           2.0      13.0
1  0.466664           0.0            1.0           5.0      12.5
2  0.285712           0.0            1.0           4.0      12.0
3  0.000000           0.0            0.0           2.0      12.0
4  0.307690           0.0            1.0           6.0      10.0

   token_set_ratio  token_sort_ratio  fuzz_ratio  fuzz_partial_ratio  \
0              100                93          93                 100
1               86                63          66                  75
2               66                66          54                  54
3               36                36          35                  40
4               67                47          46                  56

   longest_substr_ratio
0              0.982759
1              0.596154
2              0.166667
3              0.039216
4              0.175000
```

In [18]:
```
# data before preprocessing
df2.head()
```

Out[18]:
```
   id  freq_qid1  freq_qid2  q1len  q2len  q1_n_words  q2_n_words  \
0   0          1          1     66     57          14          12
1   1          4          1     51     88           8          13
2   2          1          1     73     59          14          10
```

```
3   3           1           1     50     65           11            9
4   4           3           1     76     39           13            7

   word_Common  word_Total  word_share  freq_q1+q2  freq_q1-q2
0         10.0        23.0    0.434783           2           0
1          4.0        20.0    0.200000           5           3
2          4.0        24.0    0.166667           2           0
3          0.0        19.0    0.000000           2           0
4          2.0        20.0    0.100000           4           2
```

In [19]:
```python
a=df2.as_matrix()
b=df1.as_matrix()
c=hstack([a,b,g])
```

In [20]:
```python
print "Complete dataset shape: ",c.shape
print "Dataset type as of now: ",type(c)
```

```
Complete dataset shape:  (404290, 163097)
Dataset type as of now:  <class 'scipy.sparse.coo.coo_matrix'>
```

In [21]:
```python
print "Number of features in nlp dataframe :", b.shape[1]
print "Number of features in preprocessed dataframe :", a.shape[1]
print "Number of features in question1 w2v  dataframe :", tfidfq1.shape[1]
print "Number of features in question1 w2v  dataframe :", tfidfq2.shape[1]
print "Number of features in final dataframe  :", b.shape[1]+a.shape[1]+tfidfq1.shape
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 84717
Number of features in question1 w2v  dataframe : 78351
Number of features in final dataframe  : 163097
```

In [22]:
```python
# 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 pred

    #A =(((C.T)/(C.sum(axis=1))).T)
    d = c.sum(axis=0)
    A = [c[0,0]/float(d[0]),c[0,1]/float(d[1])],[c[1,0]/float(d[0]),c[1,1]/float(d[1]

    #divid each element of the confusion matrix with the sum of elements in that colu

    # 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
# 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))
d = c.sum(axis=1)
B = [c[0,0]/float(d[0]),c[0,1]/float(d[1])],[c[1,0]/float(d[0]),c[1,1]/float(d[1]

#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
# 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=
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.title("Confusion matrix")

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

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

plt.show()
```

4. Machine Learning Models

4.1 Random train test split(70:30)

```
In [24]: X_train,X_test, y_train, y_test = train_test_split(c, y_true, stratify=y_true, test_s
```

```
In [25]: print("Number of data points in train data :",X_train.shape)
         print("Number of data points in test data :",X_test.shape)
```

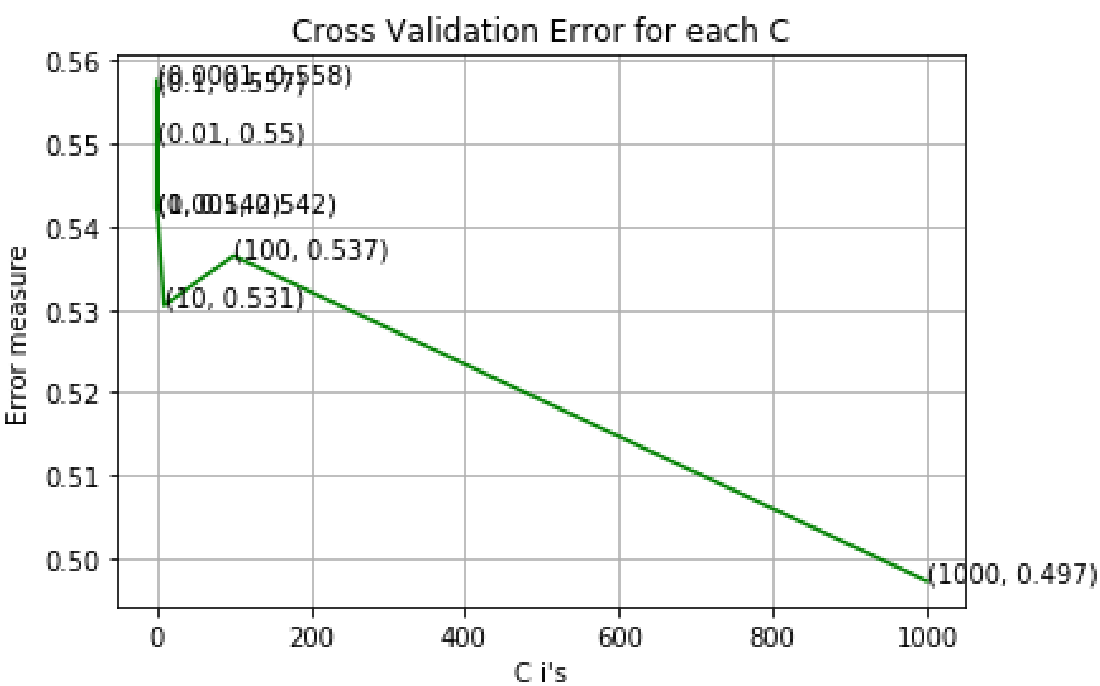
```
('Number of data points in train data :', (283003, 163097))
('Number of data points in test data :', (121287, 163097))
```

4.2 Logistic Regression with hyperparameter tuning

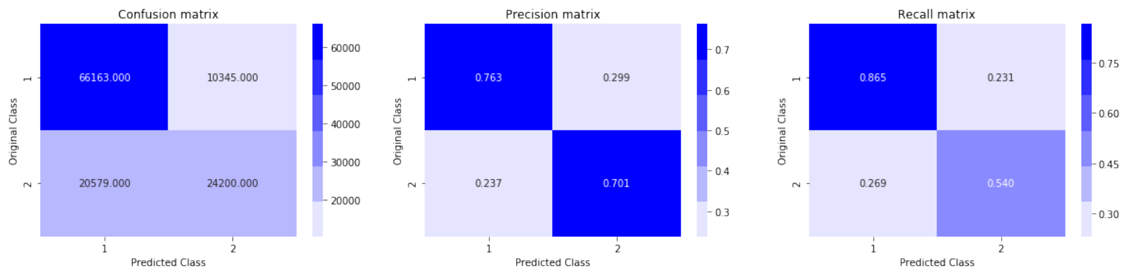
```
In [73]: #alpha = [10 ** x for x in range(-5, 4)] # hyperparam for SGD classifier.
         C = [10**(i) for i in range (-4,4)]

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # predict(X)        Predict class labels for samples in X.

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


         log_error_array=[]
         for i in C:

             #clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf=LogisticRegression(C=i, penalty='l2',random_state=5,n_jobs=-1)
             clf.fit(X_train, y_train)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(X_train, y_train)
             predict_y = sig_clf.predict_proba(X_test)
             log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
             print('For values of C = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

         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)), (C[i],log_error_array[i]))
         plt.grid()
         plt.title("Cross Validation Error for each C")
         plt.xlabel("C i's")
```

```
plt.ylabel("Error measure")
plt.show()


best_C = np.argmin(log_error_array)
#clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
clf = LogisticRegression(C=C[best_C], penalty='l2',random_state=5,n_jobs=-1)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
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 C = ', 0.0001, 'The log loss is:', 0.5576323795186202)
('For values of C = ', 0.001, 'The log loss is:', 0.541998370556302)
('For values of C = ', 0.01, 'The log loss is:', 0.5504475909755603)
('For values of C = ', 0.1, 'The log loss is:', 0.5567110450421247)
('For values of C = ', 1, 'The log loss is:', 0.5420383359131249)
('For values of C = ', 10, 'The log loss is:', 0.5305030006066515)
('For values of C = ', 100, 'The log loss is:', 0.5365015987096611)
('For values of C = ', 1000, 'The log loss is:', 0.49725385033604286)
```



```
('For values of best alpha = ', 1000, 'The train log loss is:', 0.4960970753409051)
('For values of best alpha = ', 1000, 'The test log loss is:', 0.49725385033604286)
('Total number of data points :', 121287)
```

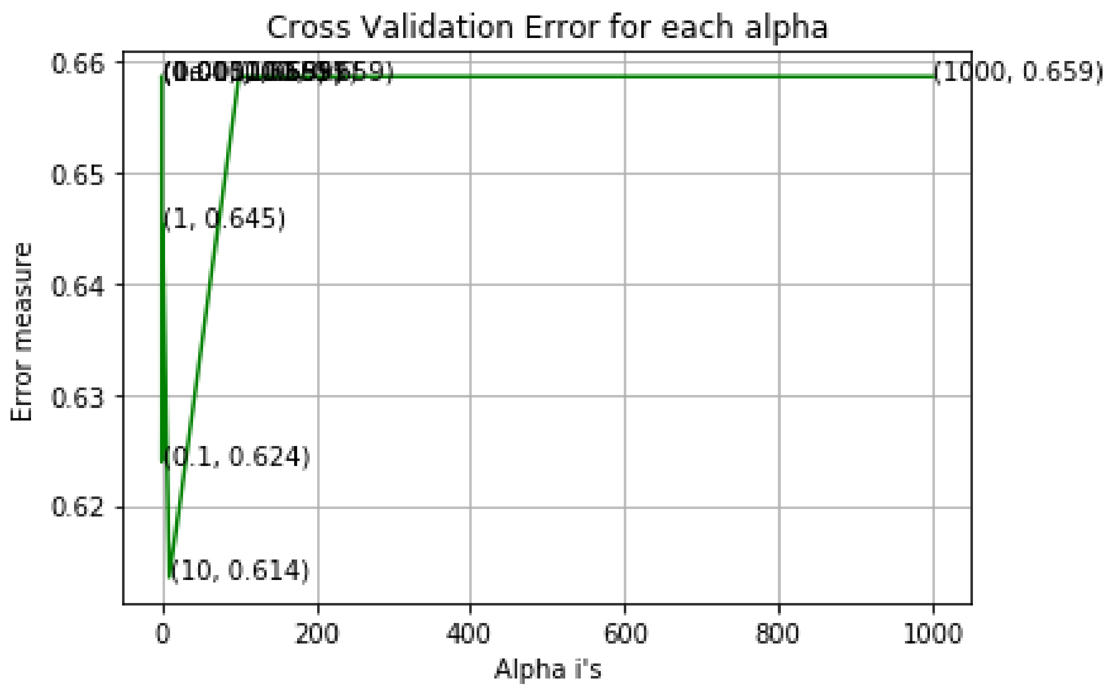


4.3 Linear SVM with hyperparameter tuning

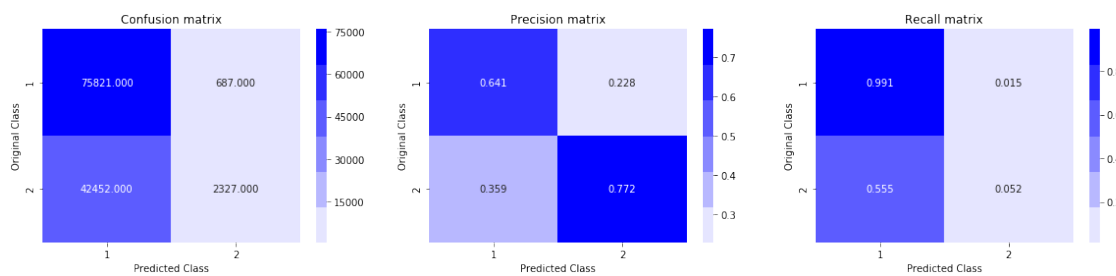
```
In [74]: alpha = [10 ** x for x in range(-5, 4)] # hyperparam for SGD classifier.

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # predict(X)        Predict class labels for samples in X.

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


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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
('For values of alpha = ', 1e-05, 'The log loss is:', 0.6585278256347589)
('For values of alpha = ', 0.0001, 'The log loss is:', 0.6585278256347589)
('For values of alpha = ', 0.001, 'The log loss is:', 0.6585278256347589)
('For values of alpha = ', 0.01, 'The log loss is:', 0.6585278256347589)
('For values of alpha = ', 0.1, 'The log loss is:', 0.6239627355084277)
('For values of alpha = ', 1, 'The log loss is:', 0.6452984658159113)
('For values of alpha = ', 10, 'The log loss is:', 0.613565435958166)
('For values of alpha = ', 100, 'The log loss is:', 0.6585278256347589)
('For values of alpha = ', 1000, 'The log loss is:', 0.6585278256347589)
```




```
('For values of best alpha = ', 10, 'The train log loss is:', 0.6130228463629108)
('For values of best alpha = ', 10, 'The test log loss is:', 0.613565435958166)
('Total number of data points :', 121287)
```



4.4 XGBoost hyperparameter tuning

```
In [38]: %%time
         from scipy.stats import uniform,randint
         import scipy.stats as st
         import xgboost as xgb
         from xgboost.sklearn import XGBClassifier
         from sklearn.model_selection import RandomizedSearchCV
```

```
params = {
    "n_estimators": st.randint(3,15),
    "max_depth": st.randint(3,20)
}

xgbreg = XGBClassifier(nthreads=-1,eta=0.02,objective='binary:logistic')
gs = RandomizedSearchCV(xgbreg, params, n_jobs=1, scoring='neg_log_loss')
gs.fit(X_train, y_train)
```

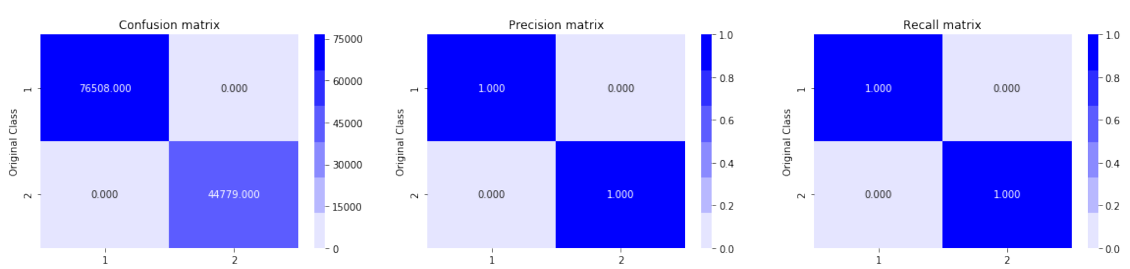
```
CPU times: user 7min 6s, sys: 28.1 s, total: 7min 34s
Wall time: 7min 34s
```

In [39]:
```
print gs.best_estimator_
print "Best n_estimators = ",gs.best_estimator_.get_params()['n_estimators']
print "Best max_depth = ",gs.best_estimator_.get_params()['max_depth']
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, eta=0.02, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=8, min_child_weight=1, missing=None,
       n_estimators=13, n_jobs=1, nthread=None, nthreads=-1,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
Best n_estimators =  13
Best max_depth =  8
```

In [40]:
```
predict_y = gs.predict_proba(X_train)
print("The train log loss is:",log_loss(y_train, predict_y, eps=1e-15))
predict_y = gs.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
('The train log loss is:', 0.1420661496112109)
('The test log loss is:', 0.1420661496163885)
('Total number of data points :', 121287)
```



1 Observation

```
In [41]: from prettytable import PrettyTable
         x = PrettyTable()

         x.field_names = ["Model Name", "Best Hyperparameter", "Train log loss", "Test log loss

         x.add_row(["Logistic Regression", "C = 1000 \n", round(0.4960970753409051,4), round(0
         x.add_row(["Linear SVM", "alpha = 10 \n", round(0.6130228463629108,4), round(0.613565
         x.add_row(["XGBoost", "n_estimators = 8 \n max_depth = 13", round(0.1420661496112109,
         print x
```

```
+---------------------+---------------------+----------------+----------------+
|      Model Name     | Best Hyperparameter | Train log loss | Test log loss |
+---------------------+---------------------+----------------+----------------+
| Logistic Regression |       C = 1000      |     0.4961     |     0.4973     |
|                     |                     |                |                |
|      Linear SVM     |      alpha = 10     |     0.613      |     0.6136     |
|                     |                     |                |                |
|       XGBoost       |  n_estimators = 8   |     0.1421     |     0.1421     |
|                     |    max_depth = 13   |                |                |
+---------------------+---------------------+----------------+----------------+
```

Out of the three models, XGBoost performed well exceptionally well with a train log loss of 0.1421 and test log loss of 0.1421. Out of three models, Linear SVM performed low compared to the Logistic Rregression and Linear SVM model with train log loss of 0.613 and test log loss of 0.1421. After hyper parameter tuning for XGBoost, the best parameters for n_estimators is 8 and max_depth is 13.

1.1 Procedure followed to solve this case study

1. Get the dataset fom kaggle competition.
2. Map the problem in general words to Machine Learning problem.
3. Perform Exploratory Data Analysis (EDA) on the accquired data.

- Read one data point and observe the features
- Observe no. of points for each class
- Observe no. of unique, repeated points for each class
- Check for null values

4. Decide some simple features that can be constructed from the given data. Here in this case, like word total, word share, frequency of total words of each question id, etc. 12 such features are identified in this case. Add these 12 features to each datapoint and perform some analysis like taking 2 features independently and trying to separate two classes based only on these featuers. This step tells how good the feature is in separting the datapoints.

5. In this step try to build some advanded features apart from the 12 constructed in the earlier step. For this to implement, first we need to preprocess the question text.

   Preprocessing of question text

- Remove HTML tags, punctuation marks
- Perform stemming
- Remove stop words
- Expanding contractions and some other preprocessing steps.

Advanced feature extraction can be donw with NLP and fuzzy features. 17 such features are constructed here. Add these 13 features to the datapoints updated in the earlier step. Like in the previous step, perform some analysis on these features like univariate analysis, bivariate analysis.

6. Plot T-SNE on this data by which we will get some idea on how well all there features help in separting the calsses.

7. This is the first step in applying Machine learning models. Remove both the questions feature from the dataset and perform TF-IDF featurization on them. This will yield some n features and append all this n features to each datapoint.

8. Apply Logistic regression with hyperparameter tuning and report train and test log loss. Apply Linear SVC with hyperparameter tuning and report train and test log loss. Apply XGBoost with hyperparameter tuning and report train and test log loss.

9. Observation part- Tabluate all these consisting model, best hyperparameter, train log loss and test log loss.