# QuoraQsPairSim

July 15, 2019

## 1 Quora Question Pair Similarity Solution

### 1.1 1. EDA

```
In [1]: 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 warnings
        warnings.filterwarnings('ignore')
```

### 1.2 1.1 Loading Data

```
In [2]: data_path = '/home/monodeepdas112/Datasets/quora-questions-sim/questions.csv'

        df = pd.read_csv(data_path)

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

```
Number of data points :  404351
```

```
In [3]: df.head()
```

```
Out[3]:    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 [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404351 entries, 0 to 404350
Data columns (total 6 columns):
id              404351 non-null int64
qid1            404351 non-null int64
qid2            404351 non-null int64
question1       404350 non-null object
question2       404349 non-null object
is_duplicate    404351 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

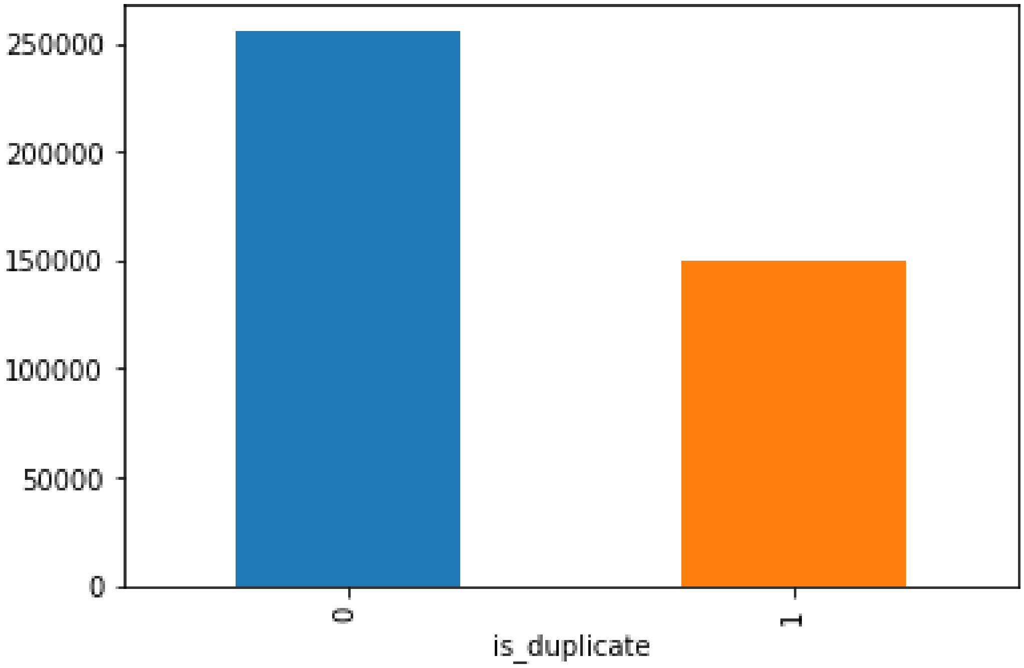
We are given a minimal number of data fields here, consisting of:

- 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.

## 1.3 1.2.1 Distribution of data points among output classes

```
In [5]: df.groupby("is_duplicate")["id"].count().plot.bar()
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f72113e3fd0>
```



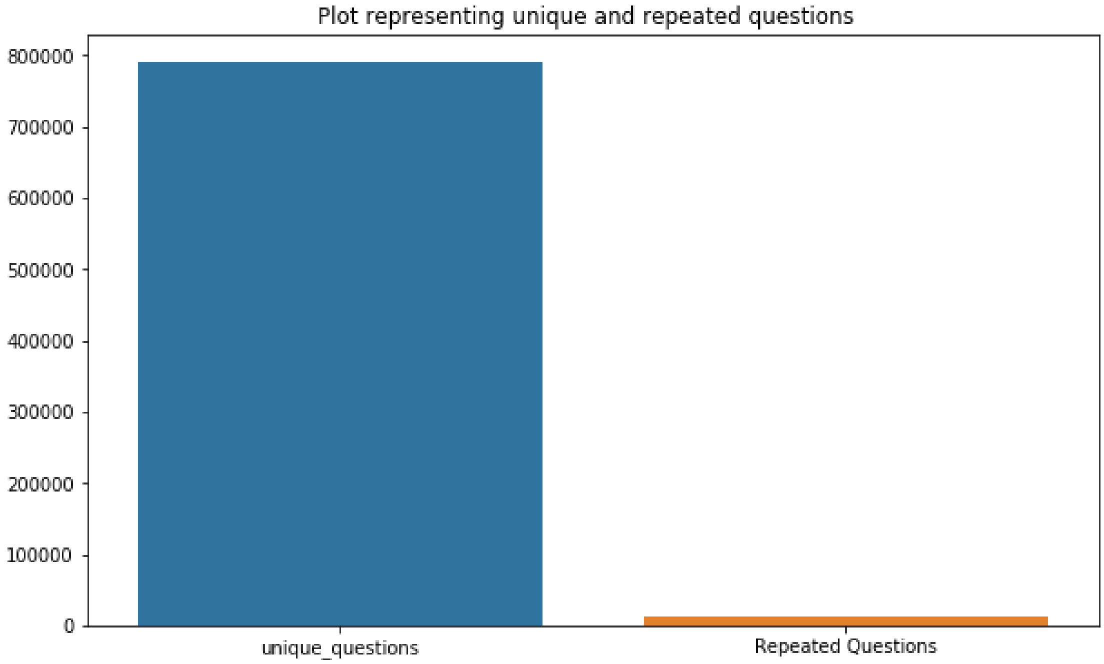
```
In [6]: print('~> Total number of question pairs for training:\n   {}'.format(len(df)))

~> Total number of question pairs for training:
   404351


In [7]: print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - roun
        print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['i

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

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

### 1.4 1.2.2 Number of unique questions

```
In [8]: 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(
        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 number of  Unique Questions are: 789801

Number of unique questions that appear more than one time: 13698 (1.7343609339567814%)

Max number of times a single question is repeated: 50
```

```
In [9]: 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()
```



## 1.5 1.2.3 Checking for Duplicates

```
In [10]: pair_uniques = df[['qid1', 'qid2', 'is_duplicate']].groupby(['qid1','qid2']).count().
In [11]: print("Number of duplicate questions", df.shape[0] - (pair_uniques).shape[0])
```

```
Number of duplicate questions 3
```

**Removing duplicates from the dataframe**

```
In [12]: df = df.drop_duplicates(subset=['qid1', 'qid2', 'is_duplicate'])
```

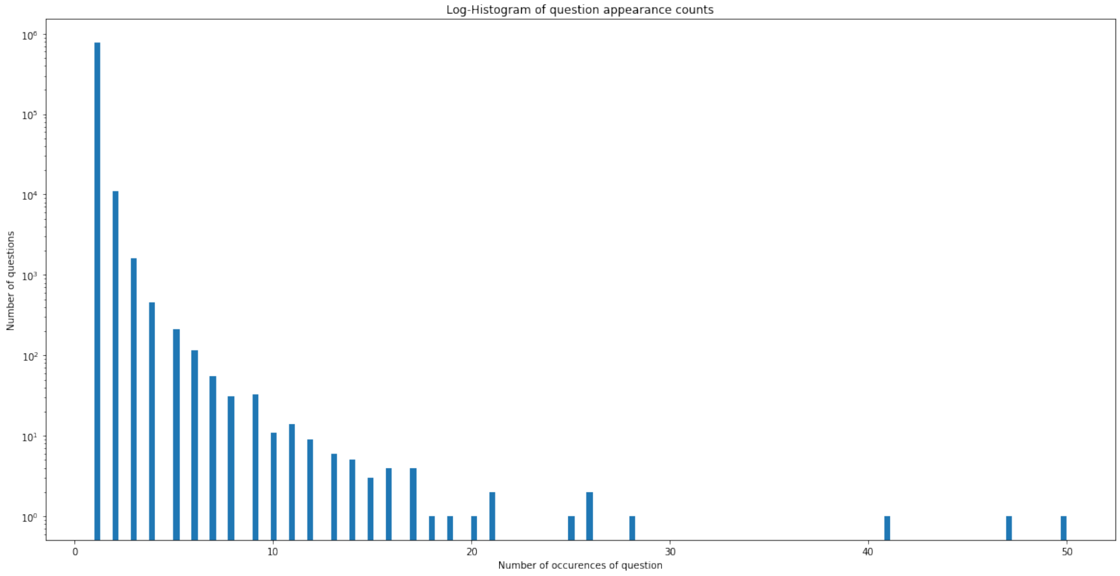
```
In [13]: print("The new number of records : ", df.shape[0])
```

```
The new number of records :  404349
```

## 1.6 1.2.4 Number of occurrences of each question

```
In [14]: 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.v
```

```
Maximum number of times a single question is repeated: 50
```



## 1.7 1.2.5 Checking for NULL values

```
In [15]: # Checking for null values in any rows
         nan_rows = df[df.isnull().any(1)]
         print(nan_rows)
```

```
            id    qid1    qid2                         question1  \
105796  105796  209841  209842    How can I develop android app?
201871  201871  398348  398349  How can I create an Android app?
363416  363416  711434  711435                               NaN

                                               question2  is_duplicate
105796                                               NaN             0
201871                                               NaN             0
363416  My Chinese name is Haichao Yu. What English na...             0
```

- There are 3 rows with null values in question2
- Filling the null values with ' '

```
In [16]: df = df.fillna('')
         nan_rows=df[df.isnull().any(1)]
         print(nan_rows)
```

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

## 1.8 1.3 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 [17]: 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)
```

```
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))

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))

data_path = '/home/monodeepdas112/Datasets/quora-questions-sim/questions_unpreprocesse

if os.path.isfile(data_path):
    df = pd.read_csv(data_path, 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(" ")))

    df['word_Common'] = df.apply(normalized_word_Common, axis=1)

    df['word_Total'] = df.apply(normalized_word_Total, axis=1)

    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(data_path, index=False)

df.head()
```

```
Out[17]:    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          1
         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          1

   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           2           0
2    0.166667           2           0
3    0.000000           2           0
4    0.100000           2           0
```

## 1.9 1.3.1 Analysis of some of the extracted features

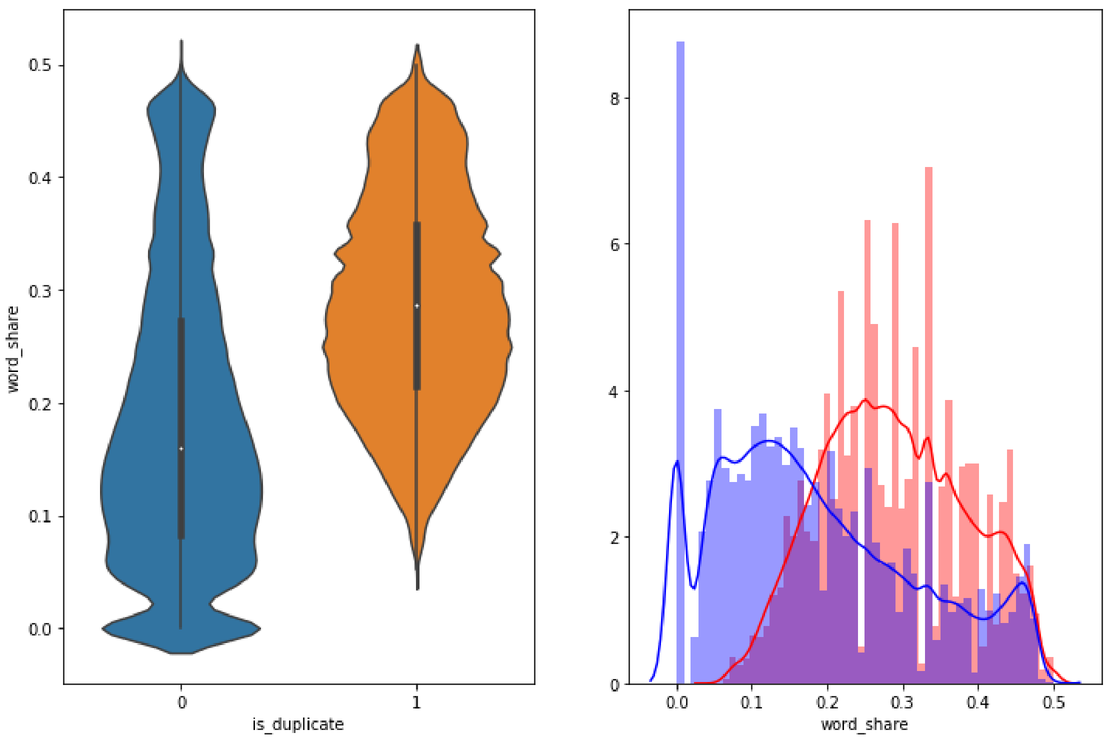
- Here are some questions have only one single words.

In [21]:
```
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']==
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']==
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 60
Number of Questions with minimum length [question2] : 25
```

### 1.3.1.1 Feature: word_share

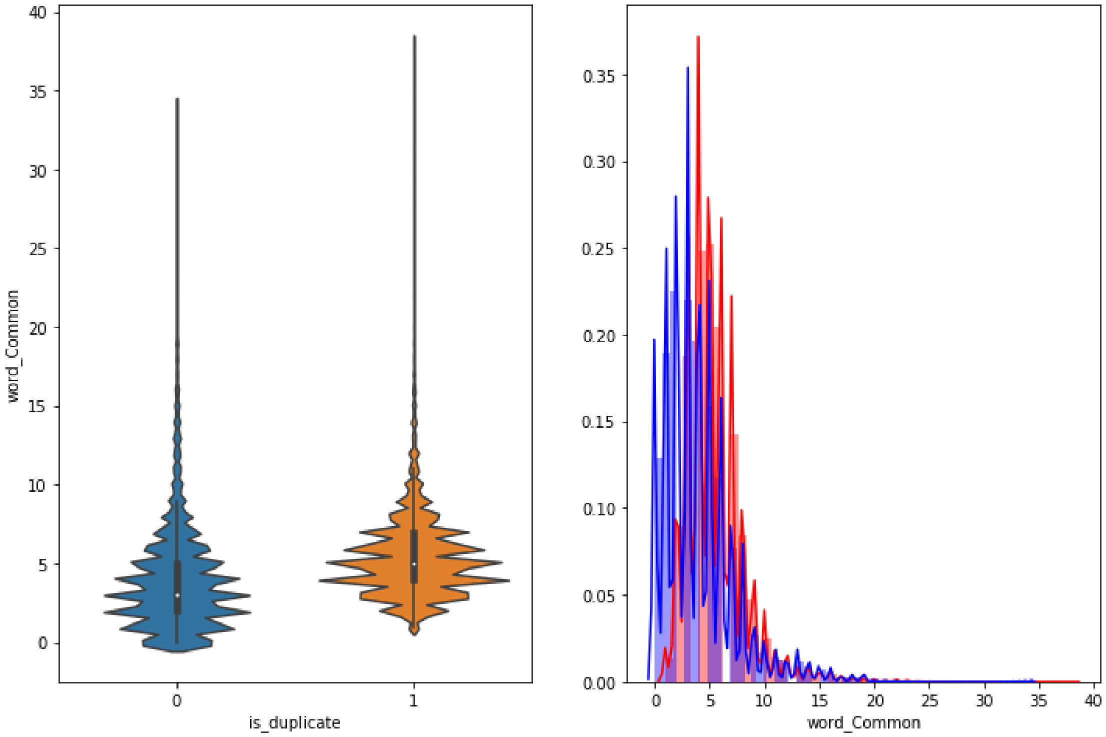
In [22]:
```
plt.figure(figsize=(12, 8))

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 = ':
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , color =
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)

#### 1.3.1.2 Feature: word_Common

In [23]:
```
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

## 1.10 1.4 EDA: Advanced Feature Extraction

```
In [2]: 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 [25]: #https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-deco
         if os.path.isfile(data_path):
             df = pd.read_csv(data_path, encoding='latin-1')
             df = df.fillna('')
             df.head()
         else:
             print("get {0} from drive or run the previous notebook".format(data_path))
```

```
In [26]: df.head(2)
```

```
Out[26]:    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          1

            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           2           0
```

### 1.10.1 1.4.1 Preprocessing of Text

- Preprocessing:
    - Removing html tags
    - Removing Punctuations
    - Performing stemming

  - Removing Stopwords
  - Expanding contractions etc.

```
In [27]: # 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("'i
                                    .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)


             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

### 1.11 1.5 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 [28]: 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_
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SA
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SA

    # 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["questio
    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-matchi
    # 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["question
    # 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 wi
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["questio
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["q
    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["ques
    return df
```

```
In [29]: data_path = '/home/monodeepdas112/Datasets/quora-questions-sim/nlp_features_train.csv
         train_csv = '/home/monodeepdas112/Datasets/quora-questions-sim/questions.csv'
         if os.path.isfile(data_path):
             df = pd.read_csv(data_path, encoding='latin-1')
             df.fillna('')
         else:
             print("Extracting features for train:")
             df = pd.read_csv(train_csv)
             df = extract_features(df)
             df.to_csv(data_path, index=False)
         df.head(2)
```

```
Out[29]:    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]
```

### 1.11.1 1.5.1 Analysis of extracted features

#### 1.5.1.1 Plotting Word clouds

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

```
In [30]: 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,
         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) : 298612
Number of data points in class 0 (non duplicate pairs) : 510090
```

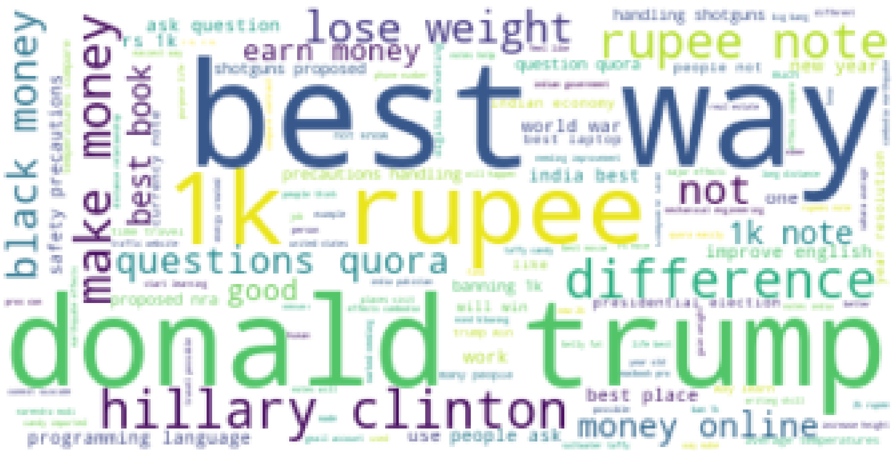
```
In [31]: # reading the text files and removing the Stop Words:
         d = path.dirname('.')

         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 : 16114225
Total number of words in non duplicate pair questions : 33201620
```

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

```
In [32]: 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
```

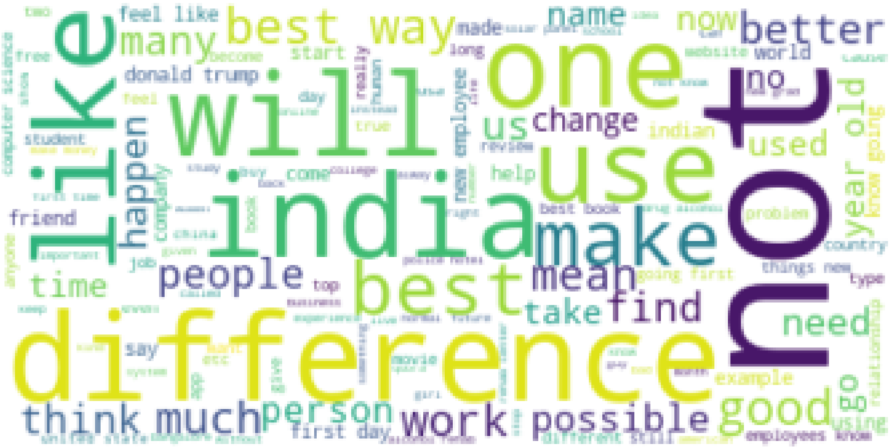


__ Word Clouds generated from non duplicate pair question's text __

```
In [33]: wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
         # generate word cloud
         wc.generate(textn_w)
```

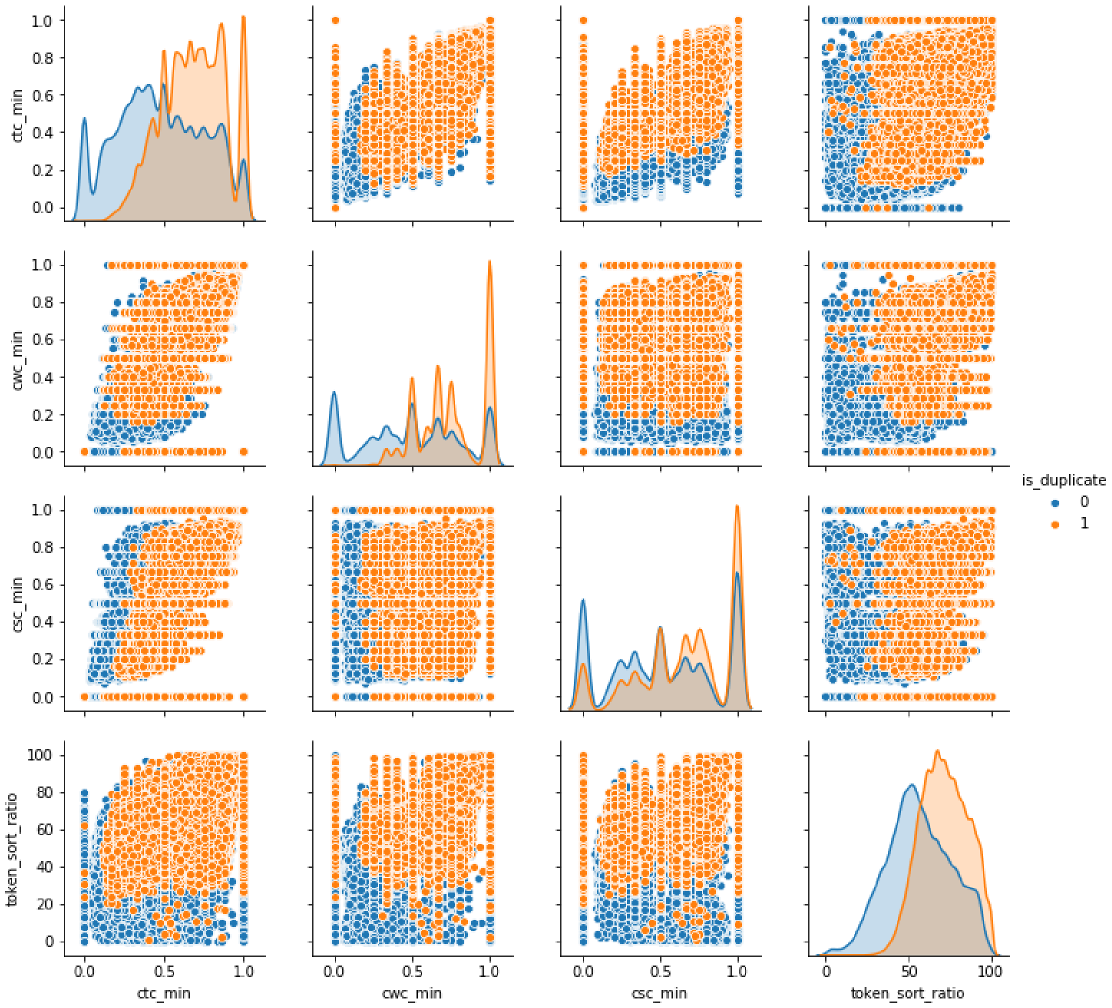
```
        print ("Word Cloud for non-Duplicate Question pairs:")
        plt.imshow(wc, interpolation='bilinear')
        plt.axis("off")
        plt.show()
```

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

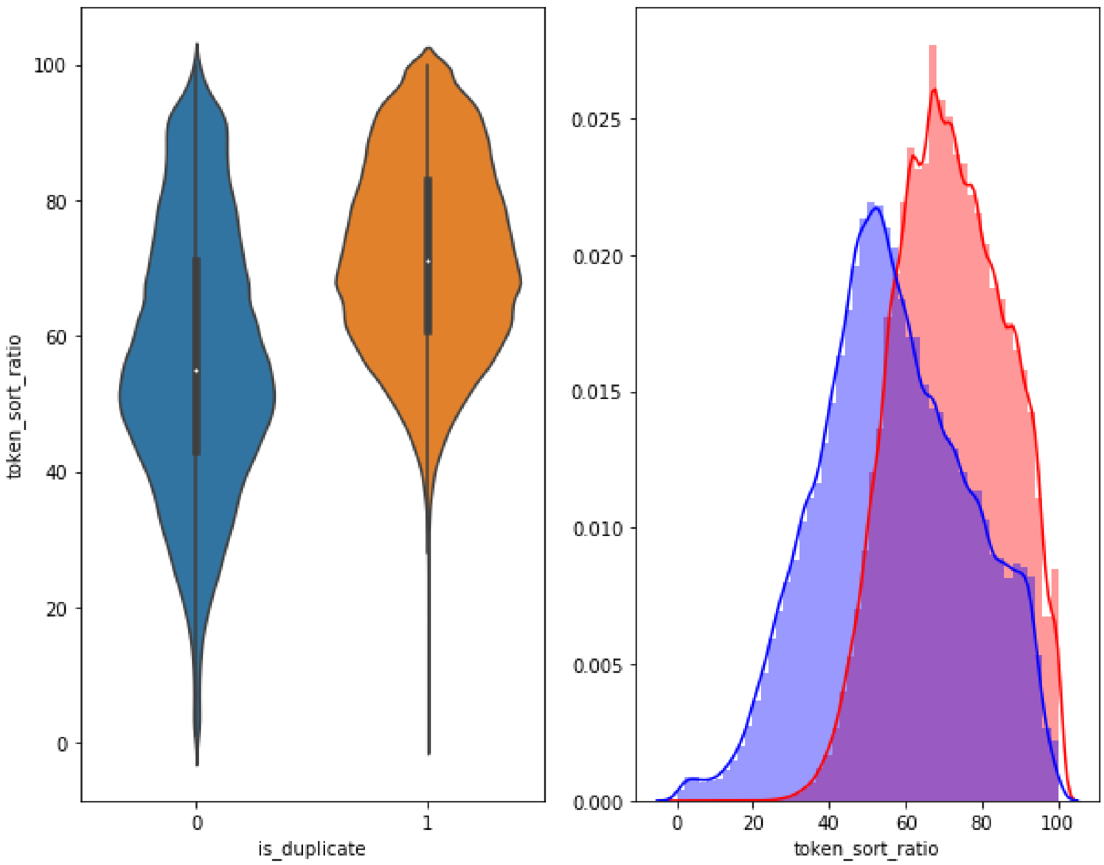


### 1.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

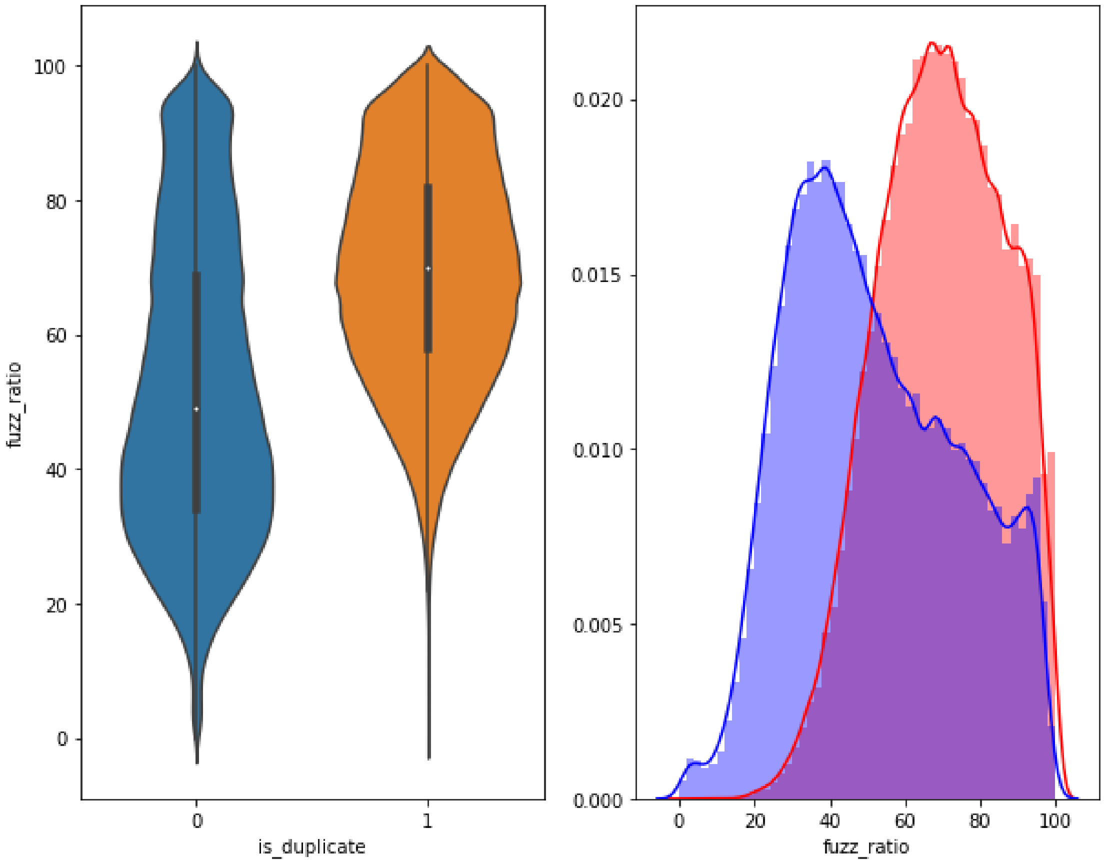
```
In [34]: n = df.shape[0]
         sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']]
         plt.show()
```



```
In [35]: # Distribution of the token_sort_ratio
         plt.figure(figsize=(10, 8))

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

         plt.subplot(1,2,2)
         sns.distplot(df[df['is_duplicate'] == 1.0]['token_sort_ratio'][0:] , label = "1", colo
         sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" , co
         plt.show()
```



```
In [36]: plt.figure(figsize=(10, 8))

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

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

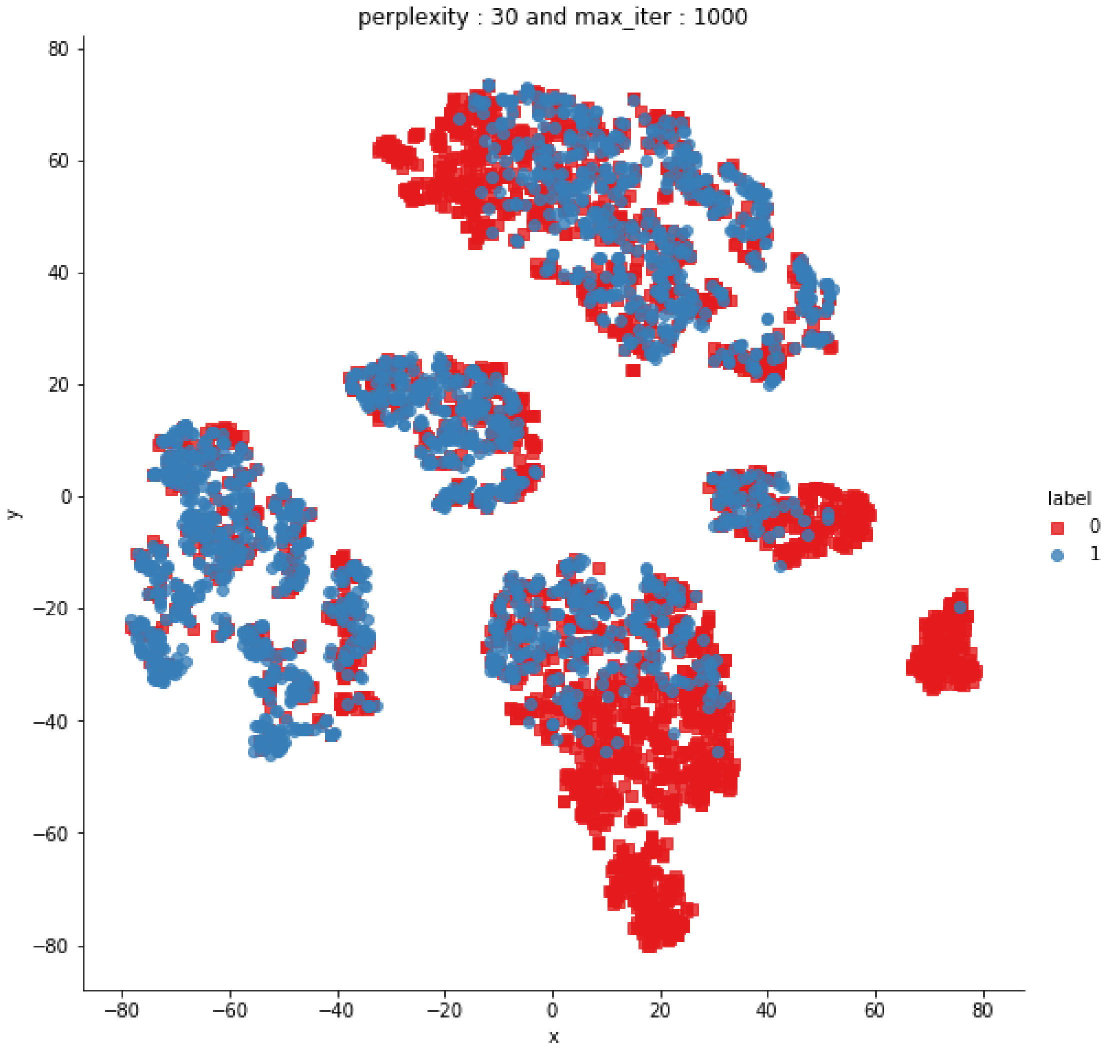


### 1.11.2 1.5.2 Visualization

```
In [37]: # Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning th

         from sklearn.preprocessing import MinMaxScaler

         dfp_subsampled = df[0:5000]
         X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'cs
         y = dfp_subsampled['is_duplicate'].values

In [38]: tsne2d = TSNE(
             n_components=2,
             init='random', # pca
             random_state=101,
             method='barnes_hut',
             n_iter=1000,
             verbose=2,
             angle=0.5
         ).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.050s...
[t-SNE] Computed neighbors for 5000 samples in 0.517s...
[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: 0.130416
[t-SNE] Computed conditional probabilities in 0.339s
[t-SNE] Iteration 50: error = 82.1324539, gradient norm = 0.0373448 (50 iterations in 3.727s)
[t-SNE] Iteration 100: error = 70.6826782, gradient norm = 0.0097551 (50 iterations in 2.404s)
[t-SNE] Iteration 150: error = 68.8895721, gradient norm = 0.0050813 (50 iterations in 2.777s)
[t-SNE] Iteration 200: error = 68.0909195, gradient norm = 0.0039069 (50 iterations in 3.306s)
[t-SNE] Iteration 250: error = 67.6059952, gradient norm = 0.0034114 (50 iterations in 3.397s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.605995
[t-SNE] Iteration 300: error = 1.7886852, gradient norm = 0.0011877 (50 iterations in 3.353s)
[t-SNE] Iteration 350: error = 1.3903050, gradient norm = 0.0004798 (50 iterations in 2.884s)
[t-SNE] Iteration 400: error = 1.2258587, gradient norm = 0.0002752 (50 iterations in 2.932s)
[t-SNE] Iteration 450: error = 1.1370399, gradient norm = 0.0001863 (50 iterations in 2.893s)
[t-SNE] Iteration 500: error = 1.0824339, gradient norm = 0.0001451 (50 iterations in 2.760s)
[t-SNE] Iteration 550: error = 1.0480006, gradient norm = 0.0001195 (50 iterations in 2.808s)
[t-SNE] Iteration 600: error = 1.0257292, gradient norm = 0.0001038 (50 iterations in 2.618s)
[t-SNE] Iteration 650: error = 1.0107807, gradient norm = 0.0000969 (50 iterations in 2.762s)
[t-SNE] Iteration 700: error = 0.9999478, gradient norm = 0.0000894 (50 iterations in 3.052s)
[t-SNE] Iteration 750: error = 0.9917220, gradient norm = 0.0000819 (50 iterations in 2.869s)
[t-SNE] Iteration 800: error = 0.9849130, gradient norm = 0.0000780 (50 iterations in 3.186s)
[t-SNE] Iteration 850: error = 0.9791487, gradient norm = 0.0000765 (50 iterations in 3.194s)
[t-SNE] Iteration 900: error = 0.9749878, gradient norm = 0.0000719 (50 iterations in 2.846s)
[t-SNE] Iteration 950: error = 0.9717448, gradient norm = 0.0000723 (50 iterations in 2.925s)
[t-SNE] Iteration 1000: error = 0.9687340, gradient norm = 0.0000687 (50 iterations in 3.040s)
[t-SNE] KL divergence after 1000 iterations: 0.968734
```

```
In [39]: 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()
```

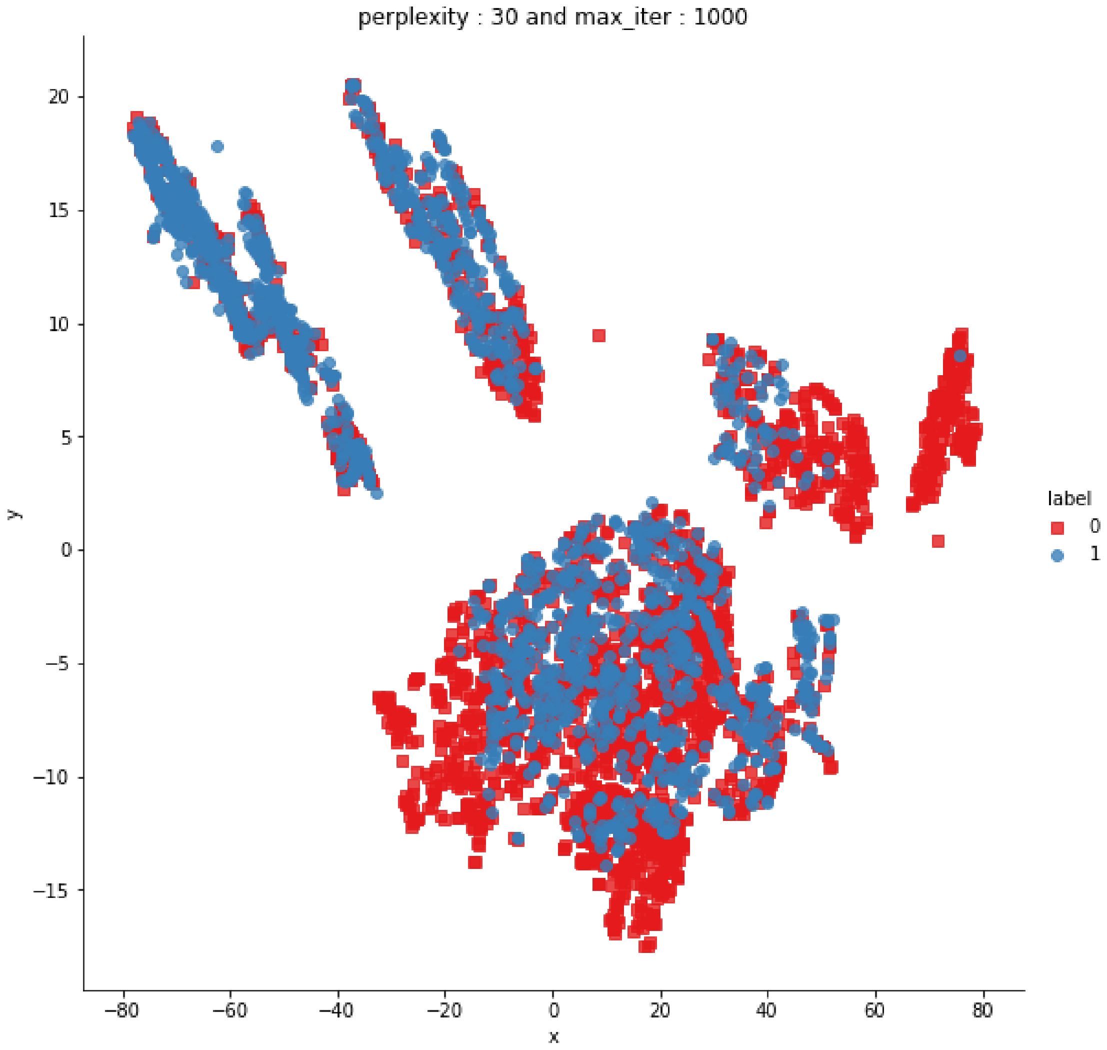


```
In [40]: from sklearn.manifold import TSNE
         tsne3d = TSNE(
             n_components=3,
             init='random', # pca
             random_state=101,
             method='barnes_hut',
             n_iter=1000,
             verbose=2,
             angle=0.5
         ).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.018s...
[t-SNE] Computed neighbors for 5000 samples in 0.598s...
[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: 0.130416
[t-SNE] Computed conditional probabilities in 0.289s
[t-SNE] Iteration 50: error = 83.4871674, gradient norm = 0.0418441 (50 iterations in 13.768s)
[t-SNE] Iteration 100: error = 69.5311432, gradient norm = 0.0037575 (50 iterations in 6.333s)
[t-SNE] Iteration 150: error = 68.0535889, gradient norm = 0.0019076 (50 iterations in 5.724s)
[t-SNE] Iteration 200: error = 67.4689713, gradient norm = 0.0012589 (50 iterations in 5.592s)
[t-SNE] Iteration 250: error = 67.1411362, gradient norm = 0.0009612 (50 iterations in 5.254s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.141136
[t-SNE] Iteration 300: error = 1.5270405, gradient norm = 0.0007029 (50 iterations in 7.760s)
[t-SNE] Iteration 350: error = 1.1922822, gradient norm = 0.0002019 (50 iterations in 10.613s)
[t-SNE] Iteration 400: error = 1.0451176, gradient norm = 0.0000971 (50 iterations in 9.624s)
[t-SNE] Iteration 450: error = 0.9719423, gradient norm = 0.0000723 (50 iterations in 8.451s)
[t-SNE] Iteration 500: error = 0.9361593, gradient norm = 0.0000553 (50 iterations in 7.680s)
[t-SNE] Iteration 550: error = 0.9186977, gradient norm = 0.0000498 (50 iterations in 8.368s)
[t-SNE] Iteration 600: error = 0.9066210, gradient norm = 0.0000431 (50 iterations in 8.593s)
[t-SNE] Iteration 650: error = 0.8959002, gradient norm = 0.0000405 (50 iterations in 7.853s)
[t-SNE] Iteration 700: error = 0.8866512, gradient norm = 0.0000375 (50 iterations in 9.033s)
[t-SNE] Iteration 750: error = 0.8798899, gradient norm = 0.0000409 (50 iterations in 8.755s)
[t-SNE] Iteration 800: error = 0.8760796, gradient norm = 0.0000332 (50 iterations in 8.586s)
[t-SNE] Iteration 850: error = 0.8727772, gradient norm = 0.0000309 (50 iterations in 7.466s)
[t-SNE] Iteration 900: error = 0.8695324, gradient norm = 0.0000307 (50 iterations in 7.982s)
[t-SNE] Iteration 950: error = 0.8659765, gradient norm = 0.0000286 (50 iterations in 7.795s)
[t-SNE] Iteration 1000: error = 0.8629071, gradient norm = 0.0000260 (50 iterations in 6.927s)
[t-SNE] KL divergence after 1000 iterations: 0.862907
```

```
In [41]: df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne3d[:,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 [42]: 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')
```

### 1.12 2 Featurizing text data with tfidf weighted word-vectors

```
In [3]: import pandas as pd
        import matplotlib.pyplot as plt
        import re
        import time
        import warnings
        import numpy as np
        from nltk.corpus import stopwords
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer
        warnings.filterwarnings("ignore")
        import sys
        import os
        import pandas as pd
        import numpy as np
        from tqdm import tqdm

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



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

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

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

```
In [64]: # avoid decoding problems
         data_path = '/home/monodeepdas112/Datasets/quora-questions-sim/questions_unpreprocess
         df = pd.read_csv(data_path)

         # 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 [65]: df.head()
```

```
Out[65]:    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          1
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          1

   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           2           0
2    0.166667           2           0
3    0.000000           2           0
4    0.100000           2           0
```

```
In [66]: # 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
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

```
In [67]: # nlp = spacy.load('en')

         # 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), len(doc1[0].vector)])
         #     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)
```

```
In [68]: # vecs2 = []
         # for qu2 in tqdm(list(df['question2'])):
         #     doc2 = nlp(qu2)
         #     mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
         #     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)
```

```
In [69]: #prepro_features_train.csv (Simple Preprocessing Feartures)
         #nlp_features_train.csv (NLP Features)
         nl_data_csv = '/home/monodeepdas112/Datasets/quora-questions-sim/nlp_features_train.c
         if os.path.isfile(nl_data_csv):
             dfnlp = pd.read_csv(nl_data_csv, encoding='latin-1')
         else:
             print("download nlp_features_train.csv from drive or run previous notebook")

         un_pre_pro_csv = '/home/monodeepdas112/Datasets/quora-questions-sim/questions_unprepr
         if os.path.isfile(un_pre_pro_csv):
             dfppro = pd.read_csv(un_pre_pro_csv, encoding='latin-1')
         else:
             print("download df_fe_without_preprocessing_train.csv from drive or run previous
In [70]: dfnlp.columns.tolist()
```

```
Out[70]: ['id',
          'qid1',
          'qid2',
          'question1',
          'question2',
          'is_duplicate',
          'cwc_min',
          'cwc_max',
          'csc_min',
          'csc_max',
          'ctc_min',
          'ctc_max',
          'last_word_eq',
          'first_word_eq',
          'abs_len_diff',
          'mean_len',
          'token_set_ratio',
          'token_sort_ratio',
          'fuzz_ratio',
          'fuzz_partial_ratio',
          'longest_substr_ratio']
```

```
In [72]: # 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 [73]: # dataframe of nlp features
         df1.head()
```

```
Out[73]:    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               63                63          43                  47
3               28                24           9                  14
4               67                47          35                  56

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

```
In [74]: # data before preprocessing
         df2.head()
```

```
Out[74]:    id  freq_qid1  freq_qid2  q1len  q2len  q1_n_words  q2_n_words  \
         0   0          1          1     66     57          14          12
         1   1          1          1     51     88           8          13
         2   2          1          1     73     59          14          10
         3   3          1          1     50     65          11           9
         4   4          1          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           2           0
         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           2           0
```

```
In [75]: # # Questions 1 tfidf weighted word2vec
         # df3_q1.head()
```

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

```
In [76]: # 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..
```

```
In [77]: # storing the final features to csv file
         final_feat = '/home/monodeepdas112/Datasets/quora-questions-sim/final_features.csv'
         if not os.path.isfile(final_feat):
             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_feat)
        else:
            result = pd.read_csv(final_feat)
```

```
In [78]: result.columns.tolist()
```

```
Out[78]: ['Unnamed: 0',
          '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',
          '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_x',
          '1_x',
          '2_x',
          '3_x',
          '4_x',
          '5_x',
          '6_x',
          '7_x',
          '8_x',
          '9_x',
          '10_x',
          '11_x',
          '12_x',
'13_x',
'14_x',
'15_x',
'16_x',
'17_x',
'18_x',
'19_x',
'20_x',
'21_x',
'22_x',
'23_x',
'24_x',
'25_x',
'26_x',
'27_x',
'28_x',
'29_x',
'30_x',
'31_x',
'32_x',
'33_x',
'34_x',
'35_x',
'36_x',
'37_x',
'38_x',
'39_x',
'40_x',
'41_x',
'42_x',
'43_x',
'44_x',
'45_x',
'46_x',
'47_x',
'48_x',
'49_x',
'50_x',
'51_x',
'52_x',
'53_x',
'54_x',
'55_x',
'56_x',
'57_x',
'58_x',
'59_x',
'60_x',
'61_x',
'62_x',
'63_x',
'64_x',
'65_x',
'66_x',
'67_x',
'68_x',
'69_x',
'70_x',
'71_x',
'72_x',
'73_x',
'74_x',
'75_x',
'76_x',
'77_x',
'78_x',
'79_x',
'80_x',
'81_x',
'82_x',
'83_x',
'84_x',
'85_x',
'86_x',
'87_x',
'88_x',
'89_x',
'90_x',
'91_x',
'92_x',
'93_x',
'94_x',
'95_x',
'0_y',
'1_y',
'2_y',
'3_y',
'4_y',
'5_y',
'6_y',
'7_y',
'8_y',
'9_y',
'10_y',
'11_y',
'12_y',
'13_y',
'14_y',
'15_y',
'16_y',
'17_y',
'18_y',
'19_y',
'20_y',
'21_y',
'22_y',
'23_y',
'24_y',
'25_y',
'26_y',
'27_y',
'28_y',
'29_y',
'30_y',
'31_y',
'32_y',
'33_y',
'34_y',
'35_y',
'36_y',
'37_y',
'38_y',
'39_y',
'40_y',
'41_y',
'42_y',
'43_y',
'44_y',
'45_y',
'46_y',
'47_y',
'48_y',
'49_y',
'50_y',
'51_y',
'52_y',
'53_y',
'54_y',
'55_y',
'56_y',
'57_y',
'58_y',
'59_y',
'60_y',
 '61_y',
 '62_y',
 '63_y',
 '64_y',
 '65_y',
 '66_y',
 '67_y',
 '68_y',
 '69_y',
 '70_y',
 '71_y',
 '72_y',
 '73_y',
 '74_y',
 '75_y',
 '76_y',
 '77_y',
 '78_y',
 '79_y',
 '80_y',
 '81_y',
 '82_y',
 '83_y',
 '84_y',
 '85_y',
 '86_y',
 '87_y',
 '88_y',
 '89_y',
 '90_y',
 '91_y',
 '92_y',
 '93_y',
 '94_y',
 '95_y']
```

### 4.2 Converting strings to numerics

```
In [79]: result.drop(result.index[0], inplace=True)
         y_true = result.is_duplicate
         result.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace=True)
```

```
In [80]: # after we read from sql table each entry was read it as a string
         # we convert all the features into numaric before we apply any model
         cols = list(result.columns)
         for i in cols:
             result[i] = result[i].apply(pd.to_numeric)
             print(i)
```

```
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
0_x
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
10_x
11_x
12_x
13_x
14_x
15_x
16_x
17_x
18_x
19_x
20_x
21_x
```

```
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
39_x
40_x
41_x
42_x
43_x
44_x
45_x
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
65_x
66_x
67_x
68_x
69_x
```

```
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
```

```
22_y
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
40_y
41_y
42_y
43_y
44_y
45_y
46_y
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
```

```
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
```

```
In [81]: # https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
         y_true = list(map(int, y_true.values))
```

4.3 Random train test split( 70:30)

```
In [82]: X_train,X_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, te
```

```
In [83]: 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 : (283045, 218)
Number of data points in test data : (121305, 218)
```

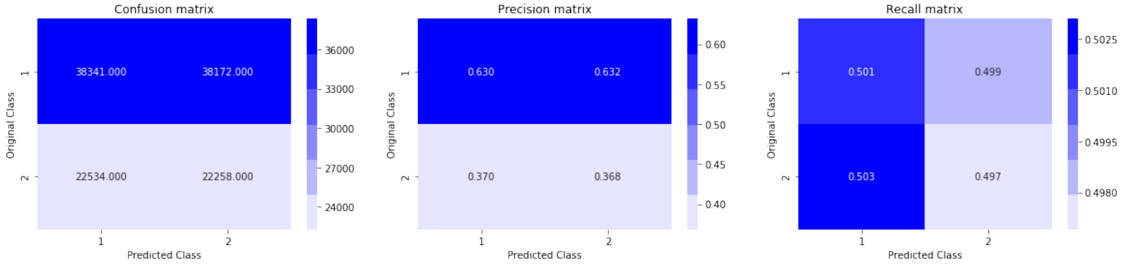
```
In [84]: print("-"*10, "Distribution of output variable in train data", "-"*10)
         train_distr = Counter(y_train)
         train_len = len(y_train)
         print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train
         print("-"*10, "Distribution of output 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:  0.6307512939638573 Class 1:  0.3692487060361427
---------- Distribution of output variable in train data ----------
Class 0:  0.369251061374222 Class 1:  0.369251061374222
```

```
In [85]: # 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)
             #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))
             #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=l
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.title("Confusion matrix")

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

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

             plt.show()
```

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

```
In [86]: # we need to generate 9 numbers and the sum of numbers should be 1
         # one solution is to genarate 9 numbers and divide each of the numbers by their sum
         # ref: https://stackoverflow.com/a/18662466/4084039
         # we create a output array that has exactly same size as the CV data
         predicted_y = np.zeros((test_len,2))
         for i in range(test_len):
             rand_probs = np.random.rand(1,2)
             predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
         print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-

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

```
Log loss on Test Data using Random Model 0.8882878002916775
```



4.5 Linear SVM with hyperparameter tuning

```
In [87]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # 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,

         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)
```

```
    ---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-87-bd687e2add60> in <module>()
     20 for i in alpha:
     21     clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
---> 22     clf.fit(X_train, y_train)
     23     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
     24     sig_clf.fit(X_train, y_train)


    ~/anaconda3/envs/AppliedAI/lib/python3.7/site-packages/sklearn/linear_model/stochastic_
    741                          loss=self.loss, learning_rate=self.learning_rate,
    742                          coef_init=coef_init, intercept_init=intercept_init,
--> 743                          sample_weight=sample_weight)
    744
    745


    ~/anaconda3/envs/AppliedAI/lib/python3.7/site-packages/sklearn/linear_model/stochastic_
    568
    569         X, y = check_X_y(X, y, 'csr', dtype=np.float64, order="C",
--> 570                          accept_large_sparse=False)
    571         n_samples, n_features = X.shape
    572


    ~/anaconda3/envs/AppliedAI/lib/python3.7/site-packages/sklearn/utils/validation.py in c
    754                     ensure_min_features=ensure_min_features,
    755                     warn_on_dtype=warn_on_dtype,
--> 756                     estimator=estimator)
    757     if multi_output:
    758         y = check_array(y, 'csr', force_all_finite=True, ensure_2d=False,


    ~/anaconda3/envs/AppliedAI/lib/python3.7/site-packages/sklearn/utils/validation.py in c
    571         if force_all_finite:
    572             _assert_all_finite(array,
--> 573                                allow_nan=force_all_finite == 'allow-nan')
    574
    575     shape_repr = _shape_repr(array.shape)


    ~/anaconda3/envs/AppliedAI/lib/python3.7/site-packages/sklearn/utils/validation.py in _
     54                 not allow_nan and not np.isfinite(X).all()):
     55             type_err = 'infinity' if allow_nan else 'NaN, infinity'
```

```
 ---> 56             raise ValueError(msg_err.format(type_err, X.dtype))
      57
      58


    ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

4.6 XGBoost

```
In [ ]: import xgboost as xgb
        params = {}
        params['objective'] = 'binary:logistic'
        params['eval_metric'] = 'logloss'
        params['eta'] = 0.02
        params['max_depth'] = 4

        d_train = xgb.DMatrix(X_train, label=y_train)
        d_test = xgb.DMatrix(X_test, label=y_test)

        watchlist = [(d_train, 'train'), (d_test, 'valid')]

        bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, verbose_eva

        xgdmat = xgb.DMatrix(X_train,y_train)
        predict_y = bst.predict(d_test)
        print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-
```

```
In [ ]: predicted_y =np.array(predict_y>0.5,dtype=int)
        print("Total number of data points :", len(predicted_y))
        plot_confusion_matrix(y_test, predicted_y)
```

5. Assignments

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

### 1.12.1 Support Vector Machines

```
In [1]: 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 warnings
        warnings.filterwarnings('ignore')
        import pprint
        import matplotlib.pyplot as plt
        import re
        import time
        import warnings
        from nltk.corpus import stopwords
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import TfidfVectorizer
        import sys
        import os
        import pandas as pd
        import numpy as np
        from tqdm import tqdm

        import pandas as pd
        import matplotlib.pyplot as plt
        import re
        import time
        import sqlite3
        from sqlalchemy import create_engine # database connection
        import csv
        import os
        warnings.filterwarnings("ignore")
        import datetime as dt
        import numpy as np
        from nltk.corpus import stopwords
        from sklearn.decomposition import TruncatedSVD
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.manifold import TSNE
        import seaborn as sns
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics.classification import accuracy_score, log_loss
        from collections import Counter
        from scipy.sparse import hstack
```

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

```
In [2]: # Loading the nlp features
        data_path = '/home/monodeepdas112/Datasets/quora-questions-sim/nlp_features_train.csv'
        data = pd.read_csv(data_path, encoding='latin-1')
```

```
In [3]: data.columns
```

```
Out[3]: Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
               'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
               'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
               'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
               'fuzz_partial_ratio', 'longest_substr_ratio'],
              dtype='object')
```

```
In [4]: data = data.dropna()
```

```
In [5]: num_data_pts = 100000
```

```
In [6]: Y = data.loc[:num_data_pts,'is_duplicate']
```

```
In [7]: X = data.loc[:num_data_pts, ['question1', 'question2', 'cwc_min', 'cwc_max', 'csc_min'
               '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']]
```

```
In [8]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(X, Y, test_size=0.30, random_s
```

### 1.12.2 [5.0.1] Defining some functions to increase code reusability and readability

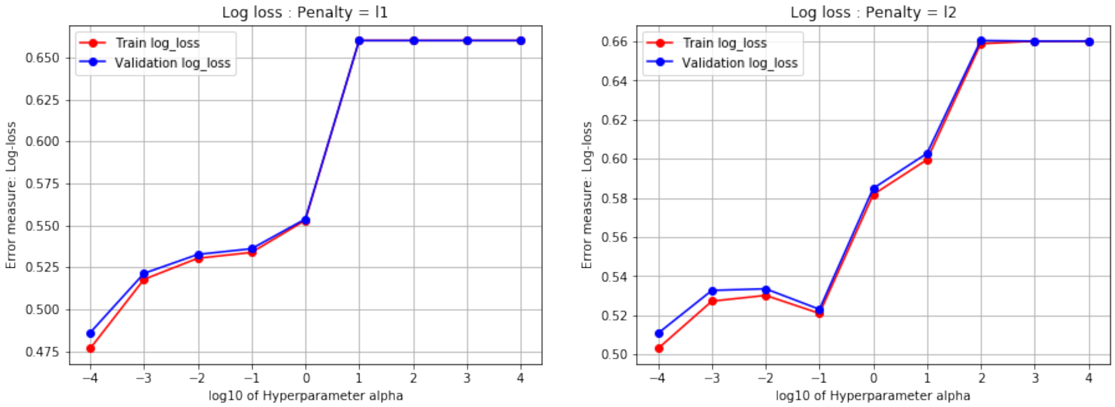
```
In [9]: class CustomVectorizer:
            def __init__(self, max_feats = None):
                if max_feats is not None:
                    self.tfidf = TfidfVectorizer(lowercase=False, max_features=max_feats)
                else:
                    self.tfidf = TfidfVectorizer(lowercase=False)

            def fit(self, X:np.array):
                b = np.vstack((X[:, :1], X[:, 1:2]))
                c = b[:, :1].tolist()
                c = [i[0] for i in c]
                self.tfidf.fit(c)
                return self

            def transform(self, X:np.array):
                q1_feats = np.array(self.tfidf.transform([i[0] for i in X[:, :1]]).todense())
                q2_feats = np.array(self.tfidf.transform([i[0] for i in X[:, 1:2]]).todense())
                nlp_feats = np.array([X[i][2:] for i in range(X.shape[0])])
                return np.hstack((nlp_feats, q1_feats, q2_feats))

In [10]: def get_vectorizer(vectorizer:str, train:np.array):
             if(vectorizer=='TFIDF'):
                 vectorizer = CustomVectorizer(max_feats=1500)
             vectorizer.fit(train)
             return vectorizer

In [11]: def perform_grid_search_cv_svm(X:pd.core.frame.DataFrame, Y:pd.core.frame.DataFrame, v

             results_path = '{0}/svm_cv_results.csv'.format(path)
             if(os.path.exists(results_path)):
                 #if present simply load the model
                 return pd.read_csv(results_path)
             else:
                 # else perform hyperparameter tuning
                 print('Performing Hyperparameter Tuning...\n')
                 # regularization parameter
                 alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
                 penalty = ['l1', 'l2']
                 hyperparameters = {
                     'svm__penalty' : penalty,
                     'svm__alpha' : alpha
                 }

                 penalties = []
                 alpha_values = []

                 train_scores = []
                 test_scores = []

                 train_mean_score = []
                 test_mean_score = []
```

```
# Initializing KFold
skf = StratifiedKFold(n_splits=3)
X = np.array(X)
Y = np.array(Y)
for reg_param in hyperparameters['svm__alpha']:
    for penalty in hyperparameters['svm__penalty']:

        #Performing Cross Validation
        for train_index, test_index in skf.split(X, Y):
            Dx_train, Dx_cv = X[train_index], X[test_index]
            Dy_train, Dy_cv = Y[train_index], Y[test_index]

            #Initializing the Vectorizer
            vectorizer = get_vectorizer('TFIDF', Dx_train)

            #Transforming the data to features
            x_train = vectorizer.transform(Dx_train)
            x_cv = vectorizer.transform(Dx_cv)

            #Initializing the LR model
            svm = SGDClassifier(penalty=penalty,
                                alpha=reg_param, loss='hinge',
                                max_iter=500, verbose=0)

            # Fit the model
            svm.fit(x_train, Dy_train)

            # Calibrating the svm model to output probablity class labels
            calib_svm = CalibratedClassifierCV(base_estimator=svm, method="iso
            calib_svm.fit(x_train, Dy_train)

            #Prediction
            train_results = calib_svm.predict_proba(x_train)
            cv_results = calib_svm.predict_proba(x_cv)

            try:
                train_score = log_loss(Dy_train, train_results[:, 1], labels=c
                test_score = log_loss(Dy_cv, cv_results[:, 1], labels=calib_sv

                #storing the results to form a dataframe
                train_scores.append(train_score)
                test_scores.append(test_score)

            except Exception as e:
                print('Error Case : ', e)
                print(('Actual, Predicted'))
                [print((Dy_cv[i], cv_results[i, 1])) for i in range(len(Dy_cv)
                             print('CV iteration : alpha={0}, penalty={1}, train_score={2}, te
                               .format(reg_param, penalty, train_score, test_score))

                         train_mean_score.append(sum(train_scores)/len(train_scores))
                         test_mean_score.append(sum(test_scores)/len(test_scores))

                         penalties.append(penalty)
                         alpha_values.append(reg_param)

                         print('C={0}, penalty={1}, train_score={2}, test_score={3}'
                               .format(reg_param, penalty, sum(train_scores)/len(train_scores)

                         train_scores = []
                         test_scores = []

                 # Creating a DataFrame from the saved data for visualization
                 results_df = pd.DataFrame({'alpha' : alpha_values, 'penalty' : penalties,
                                            'train_score' : train_mean_score,
                                            'test_score': test_mean_score})

                 #writing the results to csv after performing hyperparameter tuning
                 results_df.to_csv(results_path)

                 return results_df

In [12]: def analyse_results(df):
             # plotting error curves
             fig = plt.figure(figsize=(15, 5))
             ax = fig.gca()


             mini = df.loc[df['penalty'] == 'l1']
             plt.subplot(1, 2, 1)
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.train_score.tolist(), '-o', c='r', label='Train log_loss')
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.test_score.tolist(), '-o', c='b', label='Validation log_loss')
             plt.grid(True)
             plt.xlabel('log10 of Hyperparameter alpha')
             plt.ylabel("Error measure: Log-loss")
             plt.title('Log loss : Penalty = l1')
             plt.legend(loc='best')

             mini = df.loc[df['penalty'] == 'l2']
             plt.subplot(1, 2, 2)
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.train_score.tolist(), '-o', c='r', label='Train log_loss')
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.test_score.tolist(), '-o', c='b', label='Validation log_loss')
             plt.grid(True)
             plt.xlabel('log10 of Hyperparameter alpha')
             plt.ylabel("Error measure: Log-loss")
             plt.title('Log loss : Penalty = l2')
             plt.legend(loc='best')

             plt.show()

             # return the best parameters
             mmax = df.loc[0,'test_score']
             ind_max = 0
             for index, row in df.iterrows():
                 if(row['test_score']<mmax):
                     mmax=row['test_score']
                     ind_max = index


             best_params = {
                 'svm__alpha': df.loc[ind_max, 'alpha'],
                 'svm__penalty':df.loc[ind_max, 'penalty']
             }
             pprint.pprint(best_params)
             return best_params

In [13]: import pickle
         def retrain_svm(X, Y, best_params, vectorizer, model_path, retrain=False):
             if retrain == False:
                 if os.path.exists(model_path):
                     with open(model_path, 'rb') as input_file:
                         calib_svm = pickle.load(input_file)
                     return calib_svm
                 else:
                     raise Exception("Please retrain the model as it was not found in the give
             else:

                 X = vectorizer.transform(np.array(X))
                 Y = np.array(Y)

                 print("Retraining SVM classifier")
                 svm = SGDClassifier(penalty=best_params['svm__penalty'], alpha=best_params['s
                                     loss='hinge', max_iter=1000, verbose=0)
                 svm.fit(X, Y)

                 print("Calibrating the model")
                 calib_svm = CalibratedClassifierCV(base_estimator=svm, method="isotonic", cv=
                 calib_svm.fit(X, Y)
```

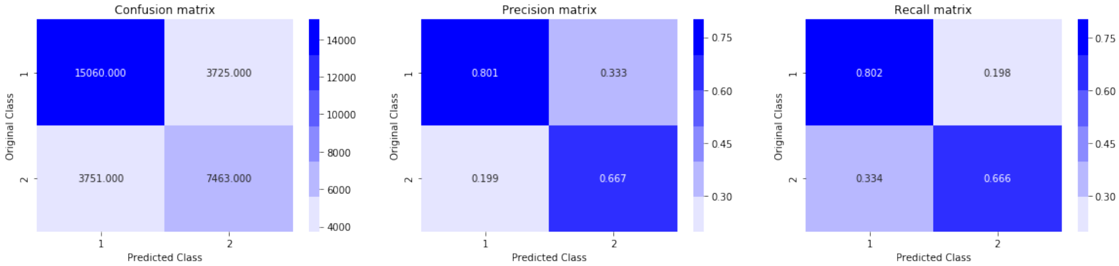
```
        # saving the trained model
        with open(model_path, 'wb') as output_file:
            pickle.dump(calib_svm, output_file)

        return calib_svm
```

```
In [14]: # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(model, X:np.array, Y:np.array):

             test_y = Y
             predict_y = model.predict(X)

             C = confusion_matrix(test_y, predict_y)

             A =(((C.T)/(C.sum(axis=1))).T)

             B =(C/C.sum(axis=0))

             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(B, 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(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.title("Recall matrix")

             plt.show()
```

```
In [15]: path = 'saved_models'
         results = perform_grid_search_cv_svm(Dx_train, Dy_train, 'TFIDF', path)
         model_path = '{0}/svm_calib_clf.pkl'.format(path)
         # Analysing results
         best_params = analyse_results(results)

         # Retraining model
         print("Retraining TFIDF vectorizer")
         vectorizer = get_vectorizer('TFIDF', np.array(Dx_train))
         clf = retrain_svm(Dx_train, Dy_train, best_params, vectorizer, model_path, False)

         # plotting confusion, precision and recall matrices
         plot_confusion_matrix(clf, vectorizer.transform(np.array(Dx_test)), np.array(Dy_test)
```



```
{'svm__alpha': 0.0001, 'svm__penalty': 'l1'}
Retraining TFIDF vectorizer
```



### 1.12.3 Logistic Regression

```
In [16]: def perform_grid_search_cv_log_reg(X:pd.core.frame.DataFrame, Y:pd.core.frame.DataFra

             results_path = '{0}/log_reg_cv_results.csv'.format(path)
             if(os.path.exists(results_path)):
```

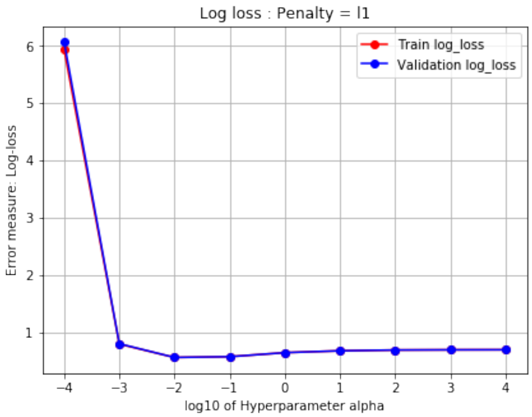
```
        #if present simply load the model
        return pd.read_csv(results_path)
    else:
        # else perform hyperparameter tuning
        print('Performing Hyperparameter Tuning...\n')
        # regularization parameter
        alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
        penalty = ['l1', 'l2']
        hyperparameters = {
            'lg__penalty' : penalty,
            'lg__alpha' : alpha
        }

        penalties = []
        alpha_values = []

        train_scores = []
        test_scores = []

        train_mean_score = []
        test_mean_score = []

        # Initializing KFold
        skf = StratifiedKFold(n_splits=3)
        X = np.array(X)
        Y = np.array(Y)
        for reg_param in hyperparameters['lg__alpha']:
            for penalty in hyperparameters['lg__penalty']:

                #Performing Cross Validation
                for train_index, test_index in skf.split(X, Y):
                    Dx_train, Dx_cv = X[train_index], X[test_index]
                    Dy_train, Dy_cv = Y[train_index], Y[test_index]

                    #Initializing the Vectorizer
                    vectorizer = get_vectorizer('TFIDF', Dx_train)

                    #Transforming the data to features
                    x_train = vectorizer.transform(Dx_train)
                    x_cv = vectorizer.transform(Dx_cv)

                    #Initializing the LR model
                    log_reg = SGDClassifier(penalty=penalty,
                                        alpha=reg_param, loss='log',
                                        max_iter=500, verbose=0)

                    # Fit the model
                    log_reg.fit(x_train, Dy_train)
```

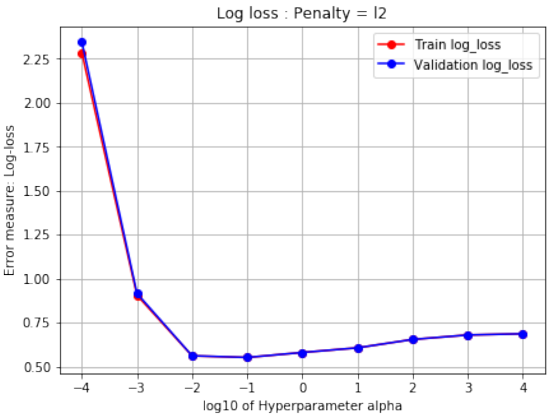
```
                             #Prediction
                             train_results = log_reg.predict_proba(x_train)
                             cv_results = log_reg.predict_proba(x_cv)

                             try:
                                 train_score = log_loss(Dy_train, train_results[:, 1], labels=l
                                 test_score = log_loss(Dy_cv, cv_results[:, 1], labels=log_reg

                                 #storing the results to form a dataframe
                                 train_scores.append(train_score)
                                 test_scores.append(test_score)

                             except Exception as e:
                                 print('Error Case : ', e)
                                 print(('Actual, Predicted'))
                                 [print((Dy_cv[i], cv_results[i, 1])) for i in range(len(Dy_cv)

                             print('CV iteration : alpha={0}, penalty={1}, train_score={2}, tes
                               .format(reg_param, penalty, train_score, test_score))

                         train_mean_score.append(sum(train_scores)/len(train_scores))
                         test_mean_score.append(sum(test_scores)/len(test_scores))

                         penalties.append(penalty)
                         alpha_values.append(reg_param)

                         print('C={0}, penalty={1}, train_score={2}, test_score={3}'
                               .format(reg_param, penalty, sum(train_scores)/len(train_scores)

                         train_scores = []
                         test_scores = []

                 # Creating a DataFrame from the saved data for visualization
                 results_df = pd.DataFrame({'alpha' : alpha_values, 'penalty' : penalties,
                                            'train_score' : train_mean_score,
                                            'test_score': test_mean_score})

                 #writing the results to csv after performing hyperparameter tuning
                 results_df.to_csv(results_path)

                 return results_df

In [17]: def analyse_results(df):
             # plotting error curves
             fig = plt.figure(figsize=(15, 5))
             ax = fig.gca()
             mini = df.loc[df['penalty'] == 'l1']
             plt.subplot(1, 2, 1)
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.train_score.tolist(), '-o', c='r', label='Train log_loss')
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.test_score.tolist(), '-o', c='b', label='Validation log_loss')
             plt.grid(True)
             plt.xlabel('log10 of Hyperparameter alpha')
             plt.ylabel("Error measure: Log-loss")
             plt.title('Log loss : Penalty = l1')
             plt.legend(loc='best')

             mini = df.loc[df['penalty'] == 'l2']
             plt.subplot(1, 2, 2)
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.train_score.tolist(), '-o', c='r', label='Train log_loss')
             plt.plot([math.log10(i) for i in mini.alpha.tolist()],
                      mini.test_score.tolist(), '-o', c='b', label='Validation log_loss')
             plt.grid(True)
             plt.xlabel('log10 of Hyperparameter alpha')
             plt.ylabel("Error measure: Log-loss")
             plt.title('Log loss : Penalty = l2')
             plt.legend(loc='best')

             plt.show()

             # return the best parameters
             mmax = df.loc[0,'test_score']
             ind_max = 0
             for index, row in df.iterrows():
                 if(row['test_score']<mmax):
                     mmax=row['test_score']
                     ind_max = index


             best_params = {
                 'lg__alpha': df.loc[ind_max, 'alpha'],
                 'lg__penalty':df.loc[ind_max, 'penalty']
             }
             pprint.pprint(best_params)
             return best_params
```

```
In [18]: import pickle
         def retrain_log_reg(X, Y, best_params, vectorizer, model_path, retrain=False):
             if retrain == False:
                 if os.path.exists(model_path):
                     with open(model_path, 'rb') as input_file:
```

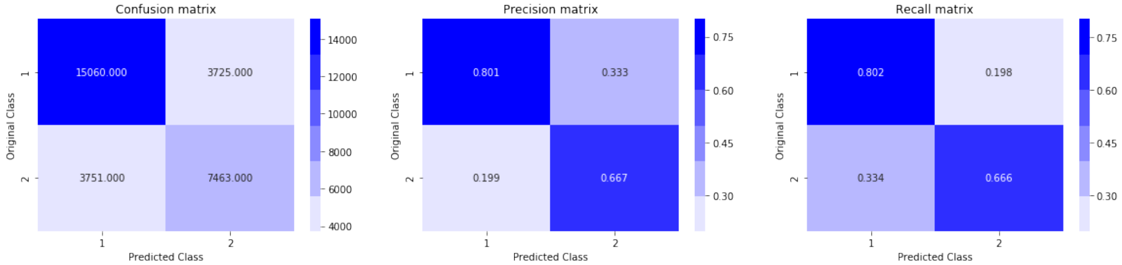
```
                log_reg = pickle.load(input_file)
            return log_reg
        else:
            raise Exception("Please retrain the model as it was not found in the given
    else:

        X = vectorizer.transform(np.array(X))
        Y = np.array(Y)

        print("Retraining SVM classifier")
        log_reg = SGDClassifier(penalty=best_params['lg__penalty'], alpha=best_params
                                loss='log', max_iter=1000, verbose=0)
        log_reg.fit(X, Y)


        # saving the trained model
        with open(model_path, 'wb') as output_file:
            pickle.dump(log_reg, output_file)

        return log_reg
```

```
In [19]: path = 'saved_models'
         results = perform_grid_search_cv_log_reg(Dx_train, Dy_train, 'TFIDF', path)
         model_path = '{0}/svm_calib_clf.pkl'.format(path)

         # Analysing results
         best_params = analyse_results(results)

         # Retraining model
         print("Retraining TFIDF vectorizer")
         vectorizer = get_vectorizer('TFIDF', np.array(Dx_train))
         clf = retrain_log_reg(Dx_train, Dy_train, best_params, vectorizer, model_path, False)

         # plotting confusion, precision and recall matrices
         plot_confusion_matrix(clf, vectorizer.transform(np.array(Dx_test)), np.array(Dy_test))
```





```
{'lg__alpha': 0.1, 'lg__penalty': 'l2'}
Retraining TFIDF vectorizer
```



### 1.12.4 XgBoost

```
In [20]: num_data_pts = 100000
```

```
In [21]: Y = data.loc[:num_data_pts,'is_duplicate']
```

```
In [22]: X = data.loc[:num_data_pts, ['question1', 'question2', 'cwc_min', 'cwc_max', 'csc_min
                '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']]
```

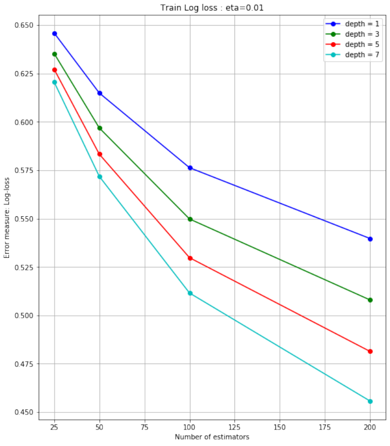
```
In [23]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(X, Y, test_size=0.30, random_
```

```
In [24]: import xgboost as xgb
         def perform_grid_search_cv_xgboost(X:pd.core.frame.DataFrame, Y:pd.core.frame.DataFra

             results_path = '{0}/xgboost_cv_results.csv'.format(path)
             if(os.path.exists(results_path)):
                 #if present simply load the model
                 return pd.read_csv(results_path)
             else:
                 # else perform hyperparameter tuning
                 print('Performing Hyperparameter Tuning...\n')

                 hyperparameters = {
                     'eta': [0.01, 0.05, 0.08, 1],
                     'max_depth': [1, 3, 5, 7],
                     'estimators': [25, 50, 100, 200]
                 }

                 etas = []
```

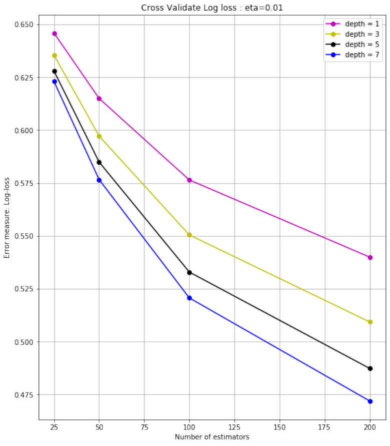
```
max_depths = []
n_estimators = []

train_scores = []
test_scores = []

train_mean_score = []
test_mean_score = []

# Initializing KFold
skf = StratifiedKFold(n_splits=3)
X = np.array(X)
Y = np.array(Y)
for eta in hyperparameters['eta']:
    for depth in hyperparameters['max_depth']:
        for estimators in hyperparameters['estimators']:

            #Performing Cross Validation
            for train_index, test_index in skf.split(X, Y):
                Dx_train, Dx_cv = X[train_index], X[test_index]
                Dy_train, Dy_cv = Y[train_index], Y[test_index]

                #Initializing the Vectorizer
                vectorizer = get_vectorizer('TFIDF', Dx_train)

                #Transforming the data to features
                Dx_train = vectorizer.transform(Dx_train)
                Dx_cv = vectorizer.transform(Dx_cv)

                bst = xgb.XGBClassifier(max_depth=depth, learning_rate=eta, n_
                                  objective='binary:logistic')

                bst.fit(Dx_train, Dy_train)

                #Prediction
                train_results = bst.predict_proba(Dx_train)
                cv_results = bst.predict_proba(Dx_cv)

                try:
                    train_score = log_loss(Dy_train, train_results[:, 1], labe
                    test_score = log_loss(Dy_cv, cv_results[:, 1], labels=bst

                    #storing the results to form a dataframe
                    train_scores.append(train_score)
                    test_scores.append(test_score)

                except Exception as e:
                    print('Error Case : ', e)
```

```
                             etas.append(eta)
                             max_depths.append(depth)
                             n_estimators.append(estimators)

                             train_mean_score.append(sum(train_scores)/len(train_scores))
                             test_mean_score.append(sum(test_scores)/len(test_scores))

                             print('eta={0}, n_estimators={1}, depth={2}, train_loss={3}, test_
                                   .format(eta, estimators, depth, sum(train_scores)/len(train_

                             train_scores = []
                             test_scores = []

                     # Creating a DataFrame from the saved data for visualization
                     results_df = pd.DataFrame({'eta' : etas, 'estimators': n_estimators, 'dep
                                                'train_score' : train_mean_score,
                                                'test_score': test_mean_score})

                     #writing the results to csv after performing hyperparameter tuning
                     results_df.to_csv(results_path)

                 return results_df

In [25]: from itertools import cycle
         cycol = cycle('bgrcmyk')
         def analyse_results(df):
             # plotting error curves
             fig = plt.figure(figsize=(20, 50))
             ax = fig.gca()
             c = 1

             eta_uniques = df['eta'].unique()
             depth_uniques = df['depth'].unique()
             estimators_uniques = df.estimators.tolist()

             for eta in eta_uniques:
                 df1 = df.query('eta=={0}'.format(eta))
                 for i in range(1,3): # train/test
                     for depth in depth_uniques:
                         mini = df1.query('depth=={0}'.format(depth))
                         x = mini.estimators.tolist()
                         plt.subplot(len(eta_uniques), 2, c)
                         plt.grid(True)
                         if i % 2 == 1:
                             plt.title('Train Log loss : eta={0}'.format(eta))
                             y = mini.train_score.tolist()
                         else:
                             plt.title('Cross Validate Log loss : eta={0}'.format(eta))
                             y = mini.test_score.tolist()
                         plt.xlabel("Number of estimators")
                         plt.ylabel("Error measure: Log-loss")
                         plt.plot(x, y, '-o', c=next(cycol), label='depth = {0}'.format(depth)
                         plt.legend(loc='best')
                     c = c + 1
             plt.show()
             mmin = df.loc[0,'test_score']
             mrow = 0
             for index, row in df.iterrows():
                 if row['test_score']<mmin:
                     mrow = index
                     mmin = row['test_score']

                     best_result = {
                         'eta': row['eta'],
                         'depth': row['depth'],
                         'estimators': row['estimators'],
                         'tr_score': row['train_score'],
                         'tst_score': mmin
                     }
             pprint.pprint(best_result)
             return best_result

In [26]: import pickle
         def retrain_xgboost(X, Y, best_params, vectorizer, model_path, retrain=False):
             if retrain == False:
                 if os.path.exists(model_path):
                     with open(model_path, 'rb') as input_file:
                         bst = pickle.load(input_file)
                     return bst
                 else:
                     raise Exception("Please retrain the model as it was not found in the give
             else:

                 X = vectorizer.transform(np.array(X))
                 Y = np.array(Y)

                 print("Retraining SVM classifier")
                 bst = xgb.XGBClassifier(max_depth=int(best_params['depth']), learning_rate=be
                                                   objective='binary:logistic')
                 bst.fit(X, Y)

                 # saving the trained model
                 with open(model_path, 'wb') as output_file:
                     pickle.dump(bst, output_file)
```

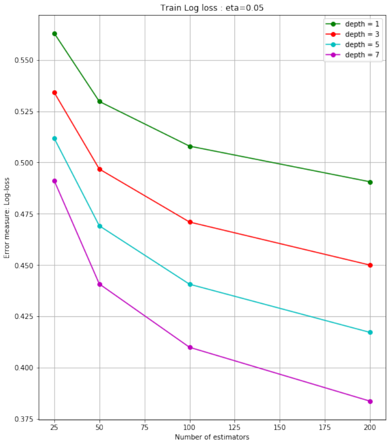
```
                return bst

In [27]: path = 'saved_models'
         results = perform_grid_search_cv_xgboost(Dx_train, Dy_train, "TFIDF", path)
         model_path = '{0}/xgboost.pkl'.format(path)

         # Analyzing results
         best_params = analyse_results(results)

         # Retraining model
         print("Retraining TFIDF vectorizer")
         vectorizer = get_vectorizer('TFIDF', np.array(Dx_train))
         clf = retrain_xgboost(Dx_train, Dy_train, best_params, vectorizer, model_path, False)

         # plotting confusion, precision and recall matrices
         plot_confusion_matrix(clf, vectorizer.transform(np.array(Dx_test)), np.array(Dy_test)
```



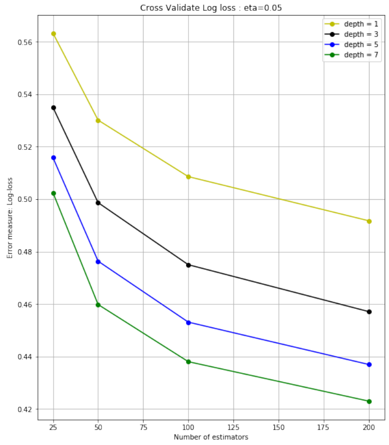
Train Log loss : eta=0.01



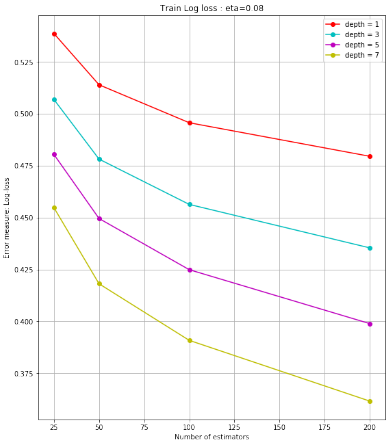
Cross Validate Log loss : eta=0.01



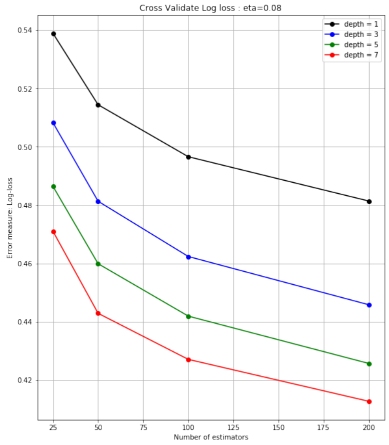
Train Log loss : eta=0.05



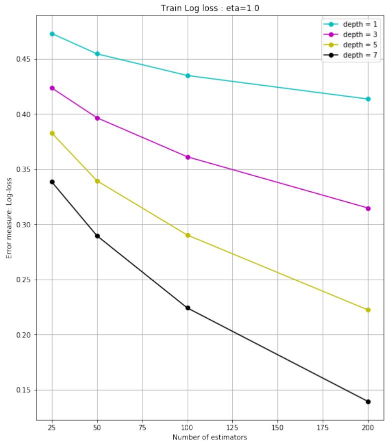
Cross Validate Log loss : eta=0.05



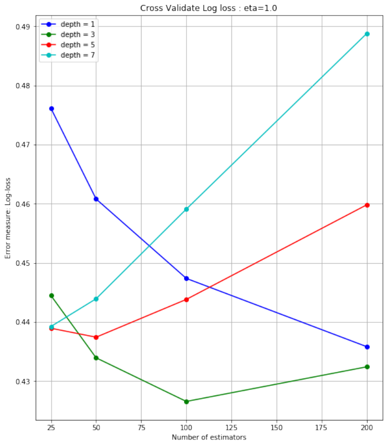
Train Log loss : eta=0.08



Cross Validate Log loss : eta=0.08

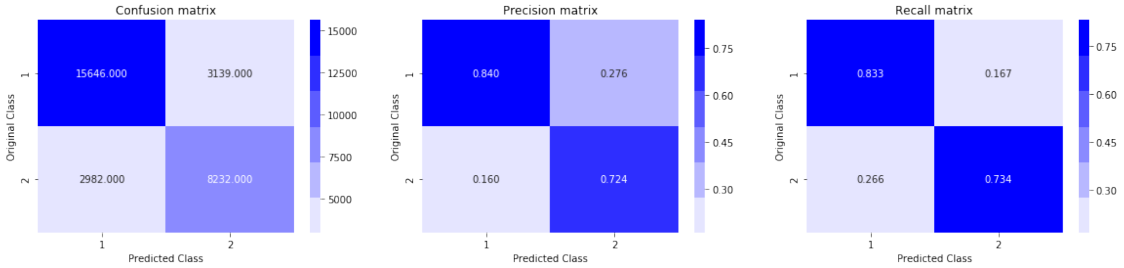


Train Log loss : eta=1.0



Cross Validate Log loss : eta=1.0

```
{'depth': 7.0,
 'estimators': 200.0,
 'eta': 0.08,
 'tr_score': 0.3615891026194618,
 'tst_score': 0.4127293757980704}
Retraining TFIDF vectorizer
```



```
In [32]: from prettytable import PrettyTable

In [37]: x = PrettyTable()

         x.field_names = ["Vectorizer", "Model", "Train log loss", "Test log loss", "%increase
         prettytable_data = [
             ["TFIDF", "SVM(alpha: 0.0001, penalty: l1", 0.476, 0.48, "{0} %".format((0.8882878
             ["TFIDF", "LogReg(alpha: 0.1, penalty: l2", 0.5514, 0.5516, "{0} %".format((0.8882
             ["TFIDF", "XGBoost(depth: 7, estimators: 200, eta: 0.08)", 0.3615891026194618, 0.4
         ]
         [x.add_row(i) for i in prettytable_data]
         print(x)
```

```
+------------+-----------------------------------------------+--------------------+------------
| Vectorizer |                     Model                     |   Train log loss   |   Test log
+------------+-----------------------------------------------+--------------------+------------
|   TFIDF    |         SVM(alpha: 0.0001, penalty: l1        |       0.476        |        0.48
|   TFIDF    |         LogReg(alpha: 0.1, penalty: l2        |       0.5514       |       0.551
|   TFIDF    | XGBoost(depth: 7, estimators: 200, eta: 0.08) | 0.3615891026194618 | 0.412729375
+------------+-----------------------------------------------+--------------------+------------
```