Quora_final

August 18, 2018

Quora Question Pairs

1. Business Problem

1.1 Description

Quora is a place to gain and share knowledge—about anything. It's a platform to ask questions and connect with people who contribute unique insights and quality answers. This empowers people to learn from each other and to better understand the world.

Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

> Credits: Kaggle

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

1.2 Sources/Useful Links

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

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.

2.2.2 Performance Metric

Source: https://www.kaggle.com/c/quora-question-pairs#evaluation

Metric(s): * log-loss : https://www.kaggle.com/wiki/LogarithmicLoss * Binary Confusion Matrix

2.3 Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

3. Exploratory Data Analysis

```
In [41]: 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
         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
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 pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import RandomizedSearchCV
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.cross_validation 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
```

3.1 Reading data and basic stats

```
In [7]: import os
        os.chdir("C:\\Users\\suman\\Downloads\\appliedaidataset\\Quora")
        df = pd.read_csv("train.csv")

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

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

```
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: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

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

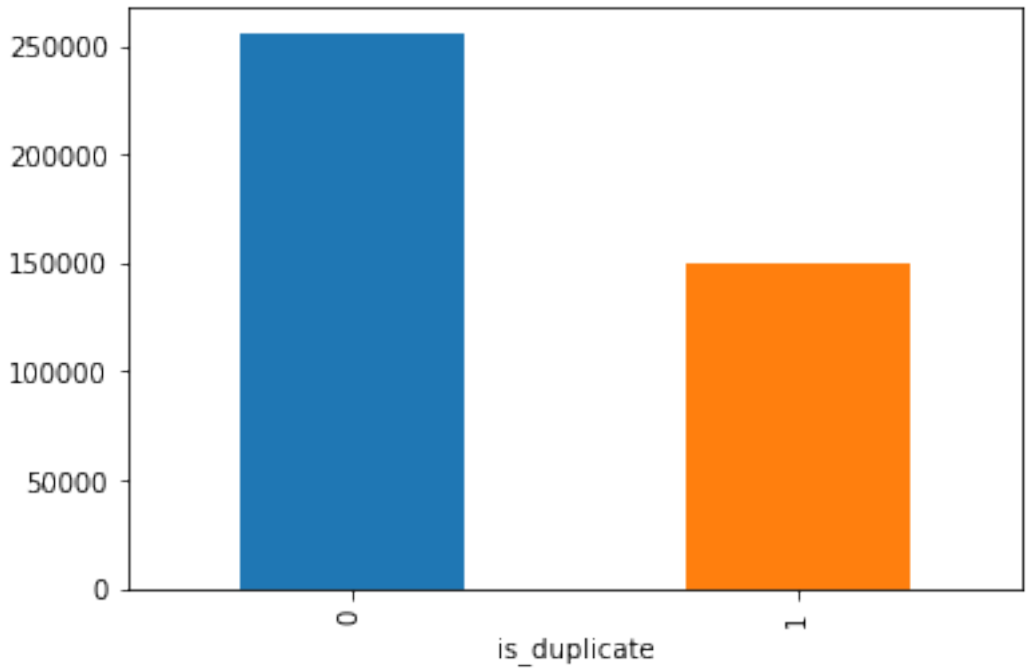
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

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

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1f25b2ba240>
```




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

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

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

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

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

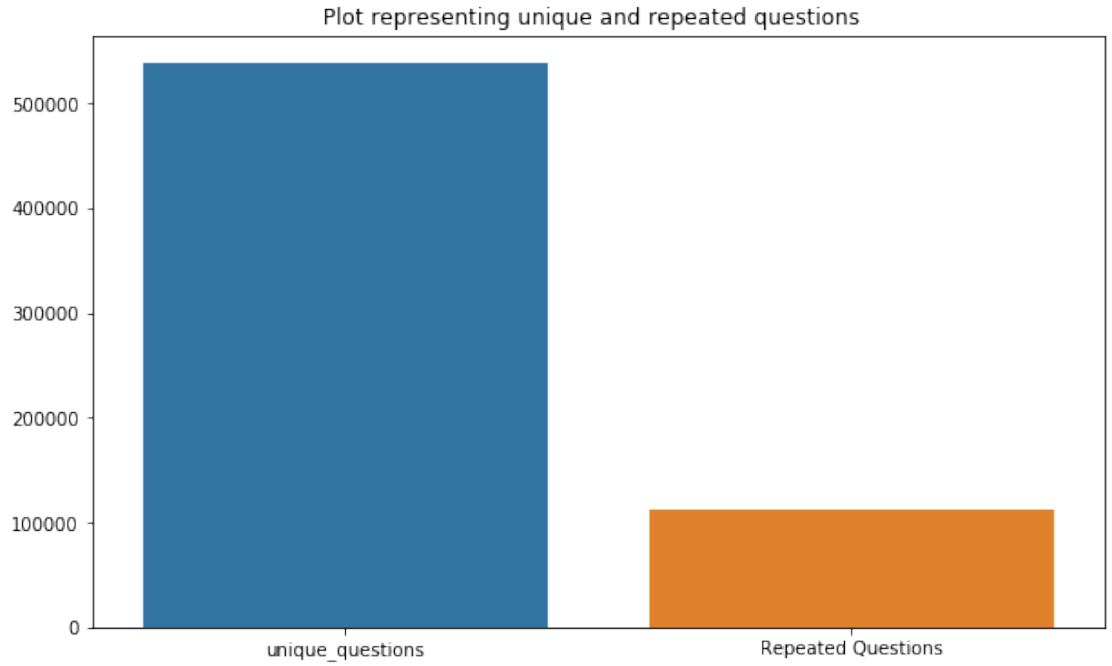
3.2.2 Number of unique questions

```
In [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: 537933

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

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

```
In [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()
```



3.2.3 Checking for Duplicates

```
In [10]: #checking whether there are any repeated pair of questions
```

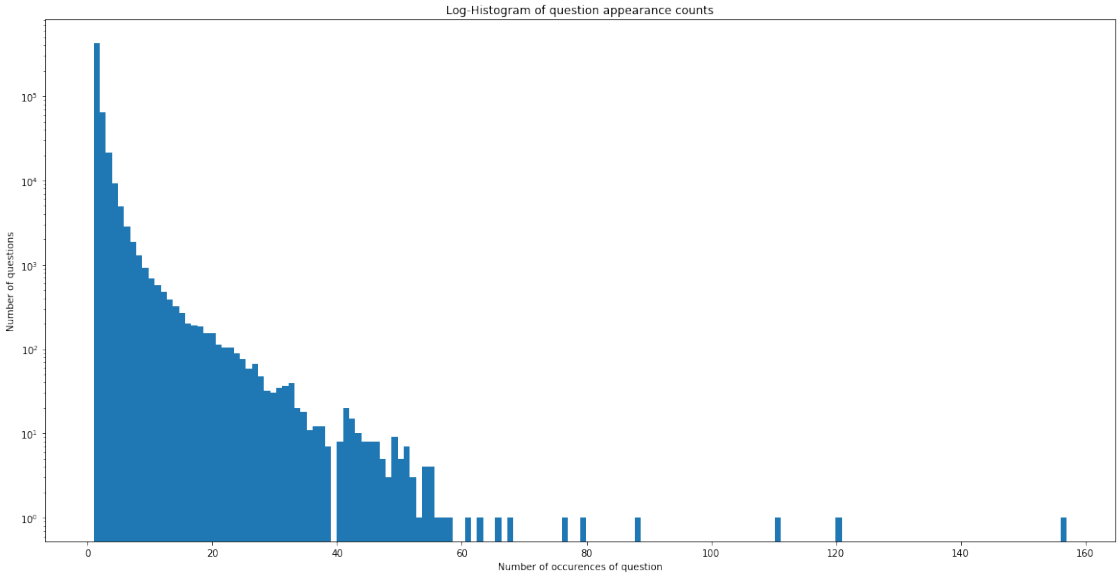
```
        pair_duplicates = df[['qid1','qid2','is_duplicate']].groupby(['qid1','qid2']).count()

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

```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

```
In [11]: 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.
```

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



3.2.5 Checking for NULL values

```
In [12]: #Checking whether there are any rows with null values
         nan_rows = df[df.isnull().any(1)]
         print (nan_rows)
```

```
            id    qid1    qid2                         question1  \
105780  105780  174363  174364    How can I develop android app?
201841  201841  303951  174364  How can I create an Android app?
363362  363362  493340  493341                               NaN

                                                 question2  is_duplicate
105780                                                 NaN             0
201841                                                 NaN             0
363362  My Chinese name is Haichao Yu. What English na...             0
```

- There are 2 rows with null values in question2 and one in question1

```
In [14]: # Filling the null values with ' '
         df = df.fillna('')
         nan_rows = df[df.isnull().any(1)]
         print (nan_rows)
```

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like: - ____freq_qid1____ = Frequency of qid1's - ____freq_qid2____ = Frequency of qid2's - ____q1len____ = Length of q1 - ____q2len____ = Length of q2 - ____q1_n_words____ = Number of words in Question 1 - ____q2_n_words____ = Number of words in Question 2 - ____word_Common____ = (Number of common unique words in Question 1 and Question 2) - ____word_Total____ =(Total num of words in Question 1 + Total num of words in Question 2) - ____word_share____ = (word_common)/(word_Total) - ____freq_q1+freq_q2____ = sum total of frequency of qid1 and qid2 - ____freq_q1-freq_q2____ = absolute difference of frequency of qid1 and qid2

```
In [15]: if os.path.isfile('df_fe_without_preprocessing_train.csv'):
             df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
         else:
             df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
             df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
             df['q1len'] = df['question1'].str.len()
             df['q2len'] = df['question2'].str.len()
             df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
             df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

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

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

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

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

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

df.head()
```

Out[15]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	freq_q1-q2
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14	12	10.0	23.0	0.434783	2	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	8	13	4.0	20.0	0.200000	5	3
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	14	10	4.0	24.0	0.166667	2	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65	11	9	0.0	19.0	0.000000	2	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	13	7	2.0	20.0	0.100000	4	2

3.3.1 Analysis of some of the extracted features

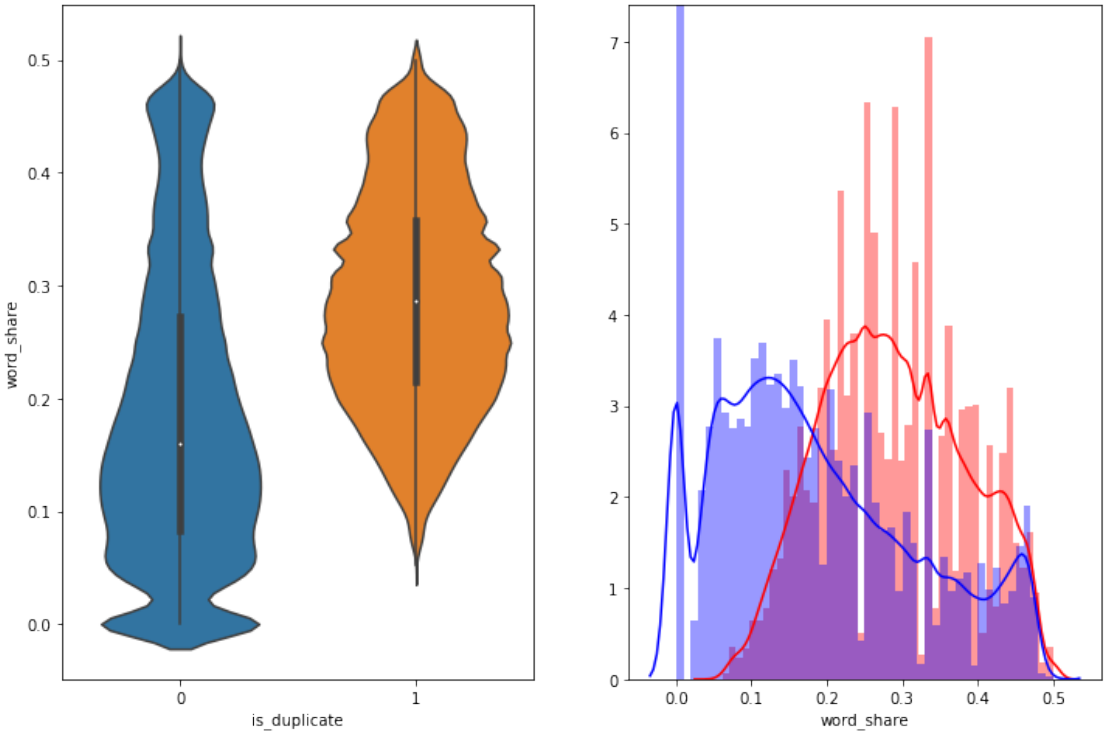
- Here are some questions have only one single words.

```
In [16]: 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] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

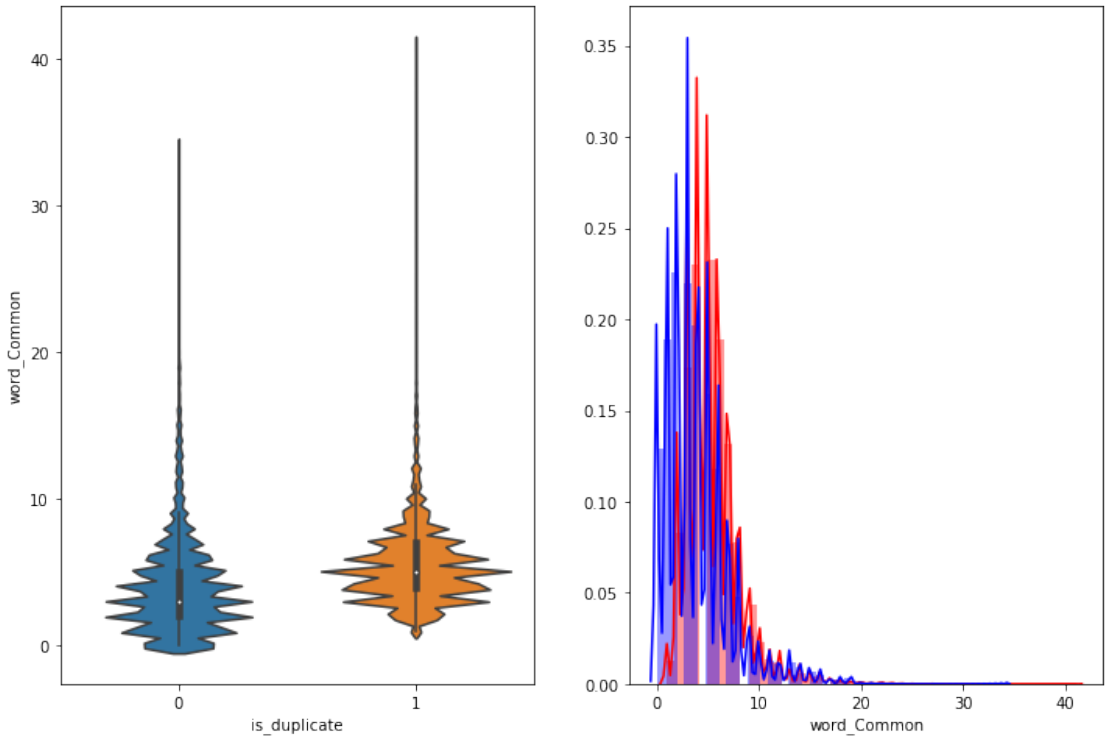
```
In [18]: import warnings
         warnings.filterwarnings("ignore")
         plt.figure(figsize=(12, 8))

         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)

3.3.1.2 Feature: word_Common

```
In [19]: 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

```
In [3]: if os.path.isfile('df_fe_without_preprocessing_train.csv'):
            df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='utf8') #'latin-
```

```
        df = df.fillna('')
        df.head()
    else:
        print("get df_fe_without_preprocessing_train.csv from drive or run the previous not
```

```
get df_fe_without_preprocessing_train.csv from drive or run the previous notebook
```

3.4 Preprocessing of Text

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

```
In [4]: # 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("'re
                                    .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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

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

Features: - **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2 cwc_min = common_word_count / (min(len(q1_words), len(q2_words)) - **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2 cwc_max = common_word_count / (max(len(q1_words), len(q2_words)) - **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2 csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops)) - **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops)) - **ctc_min** : Ratio of common_token_count to min lenghth of token count of Q1 and Q2ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))

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

```
In [5]: 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["question2

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

        #Computing Fuzzy Features and Merging with Dataset

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

        df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"
        # The token sort approach involves tokenizing the string in question, sorting the
        # then joining them back into a string We then compare the transformed strings wit
        df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question
        df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["que
        df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"]
        df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["questi
        return df
```

In [8]:
```
#pip install distance from anaconda prompt
import distance
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
```

```
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[8]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	token_set_ratio	token_sort_ratio	fuzz_ratio	fuzz_partial_ratio	longest_substr_ratio
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983	...	0.785709	0.0	1.0	2.0	13.0	100	93	93	100	0.982759
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988	...	0.466664	0.0	1.0	5.0	12.5	86	63	66	75	0.596154

[2 rows x 21 columns]

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

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

```
In [10]: 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) : 298526
Number of data points in class 0 (non duplicate pairs) : 510054
```

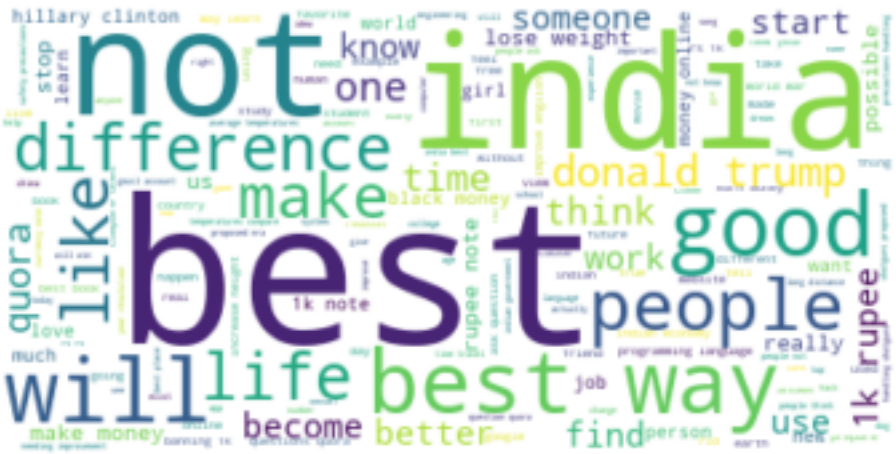
```
In [11]: # reading the text files and removing the Stop Words:
         d = path.dirname('.')

         textp_w = open(path.join(d, 'train_p.txt'),encoding="utf8").read()
         textn_w = open(path.join(d, 'train_n.txt'),encoding="utf8").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 : 891343
Total number of words in non duplicate pair questions : 33193130
```

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

```
In [12]: 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
```



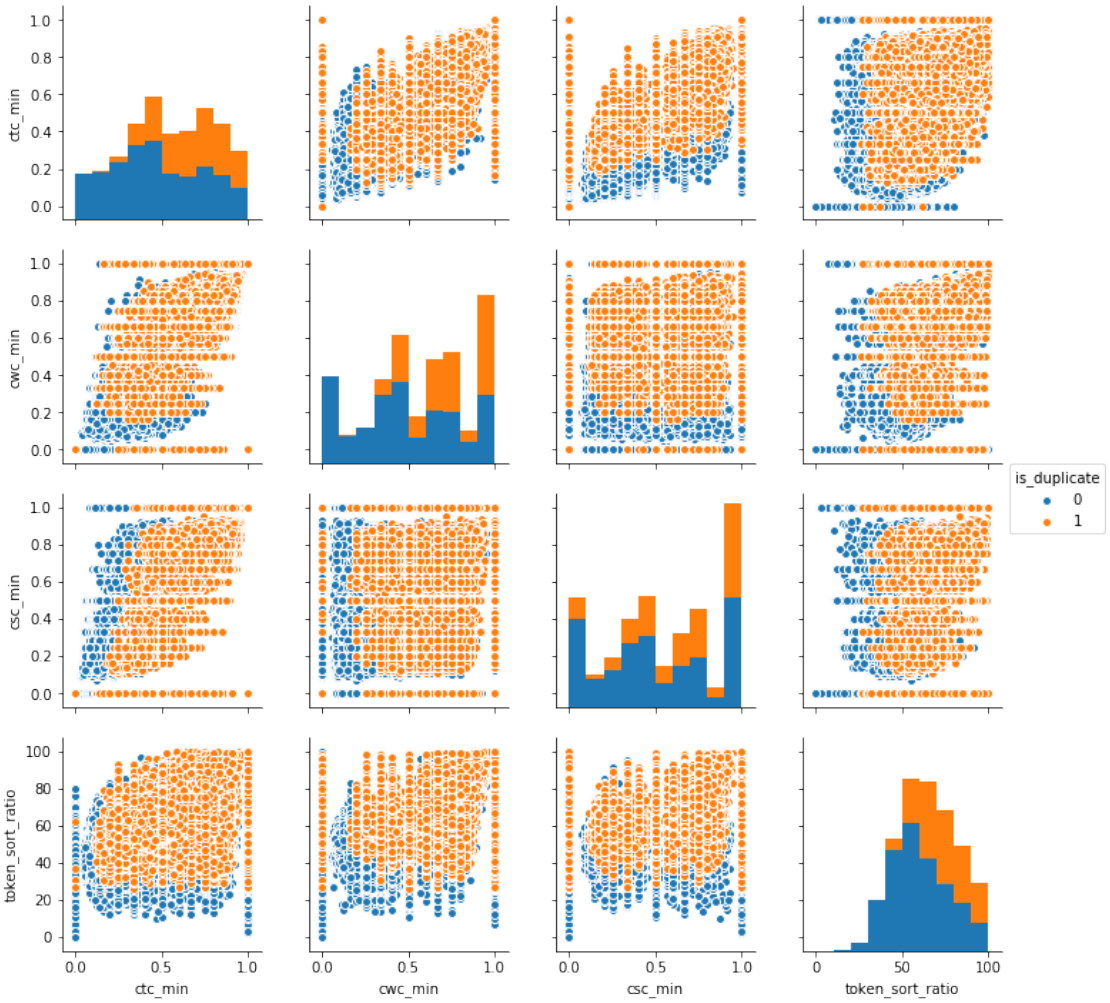
__ Word Clouds generated from non duplicate pair question's text __

```
In [13]: wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
         # generate word cloud
         wc.generate(textn_w)
         print ("Word Cloud for non-Duplicate Question pairs:")
         plt.imshow(wc, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```

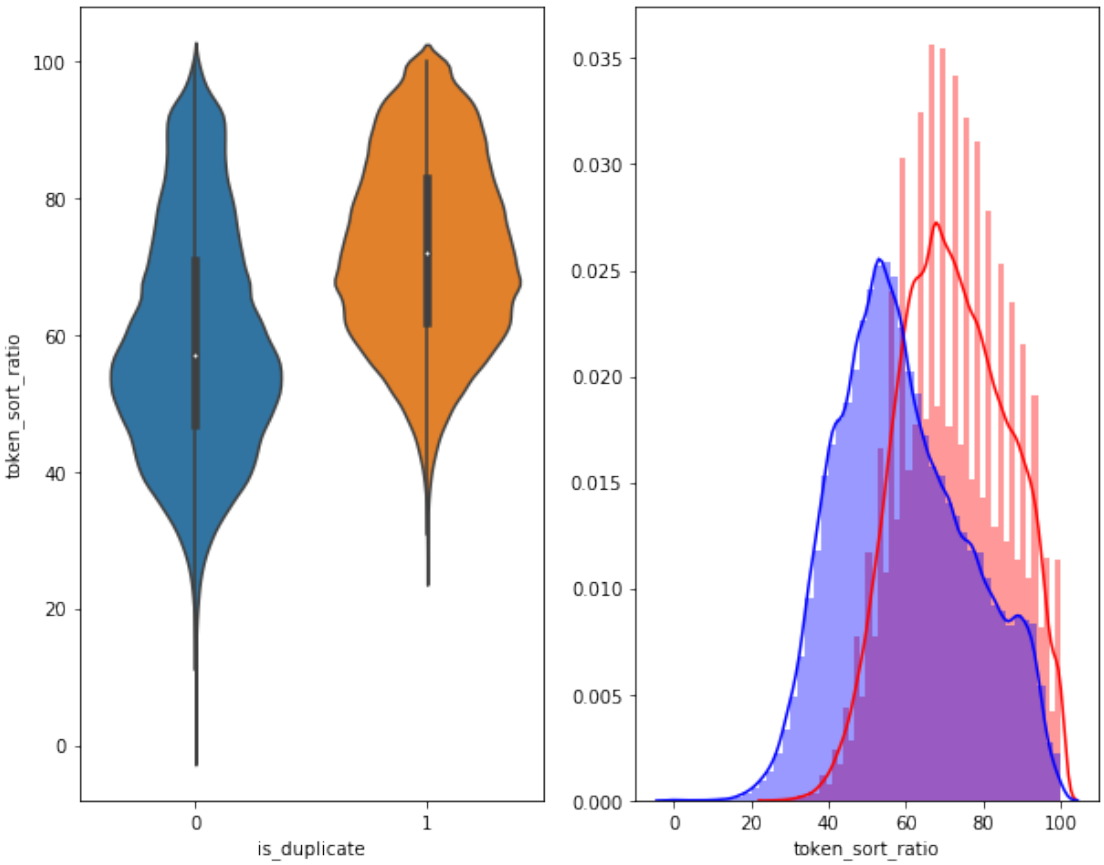
```
Word Cloud for non-Duplicate Question pairs:
```

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

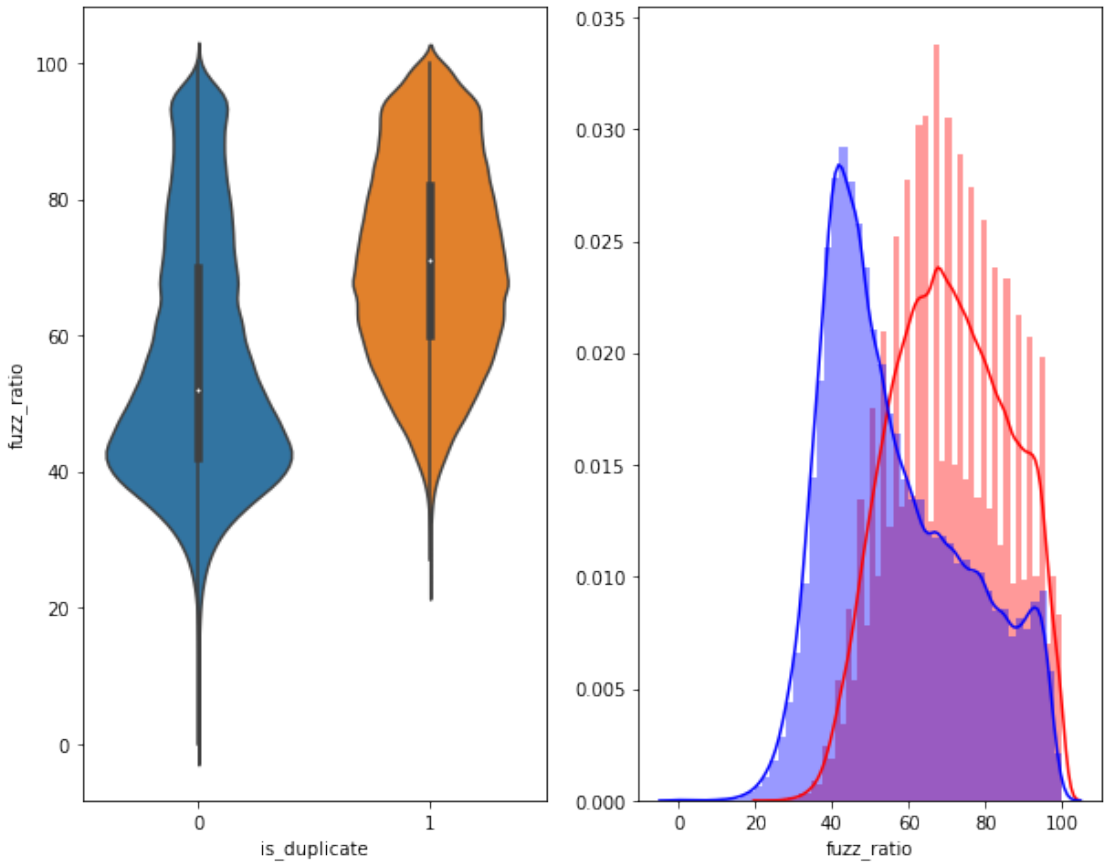
In [14]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']]
plt.show()
```



In [15]:
```
# Distribution of the token_sort_ratio
warnings.filterwarnings("ignore")
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" , col
plt.show()
```




```
In [16]: warnings.filterwarnings("ignore")

         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()
```



3.5.2 Visualization

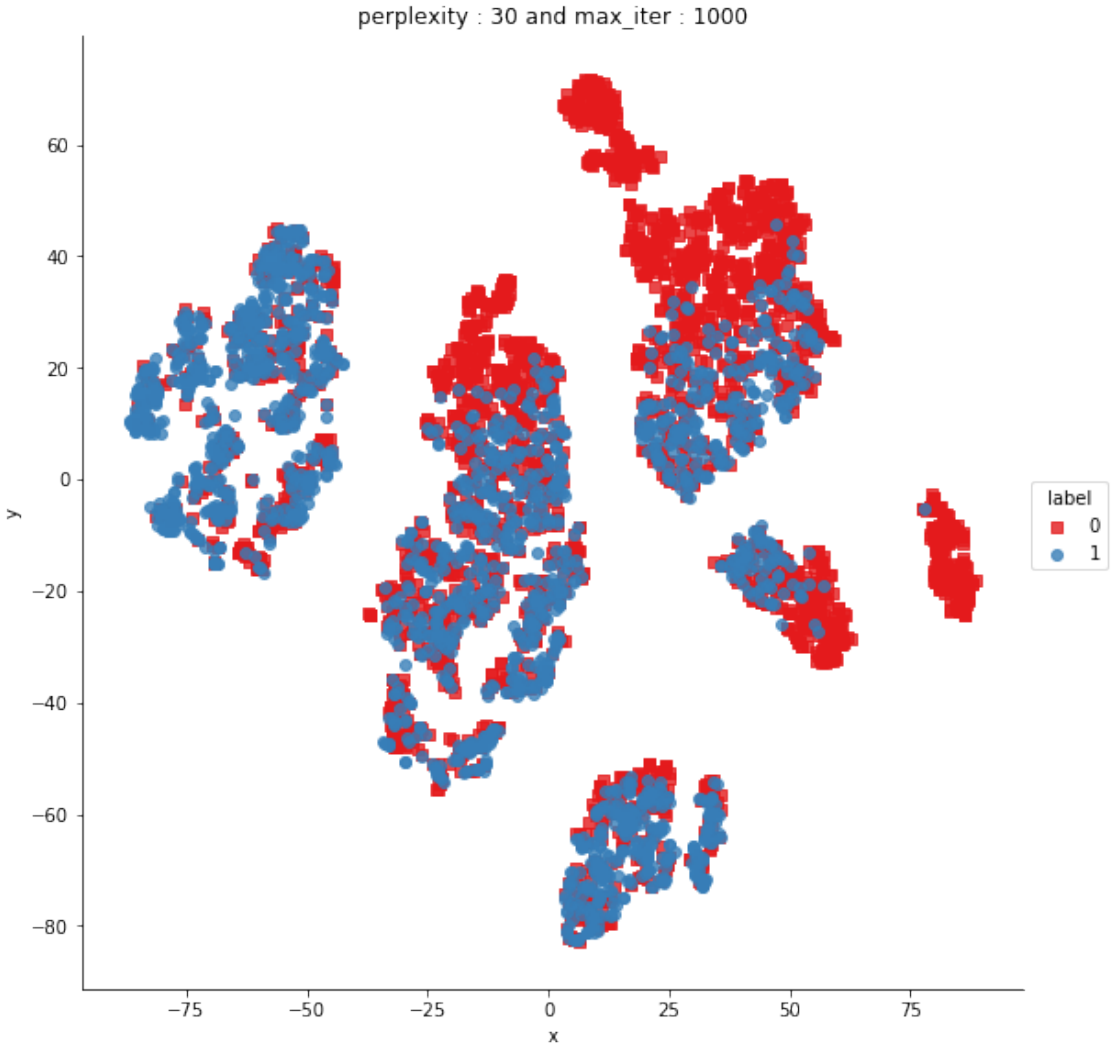
```
In [17]: # Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning th

         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 [18]: 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 0.072s...
```

```
[t-SNE] Computed neighbors for 5000 samples in 0.662s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.441s
[t-SNE] Iteration 50: error = 80.8968964, gradient norm = 0.0430571 (50 iterations in 13.351s)
[t-SNE] Iteration 100: error = 70.3833160, gradient norm = 0.0099593 (50 iterations in 10.110s)
[t-SNE] Iteration 150: error = 68.6159134, gradient norm = 0.0056708 (50 iterations in 11.511s)
[t-SNE] Iteration 200: error = 67.7694321, gradient norm = 0.0040581 (50 iterations in 9.791s)
[t-SNE] Iteration 250: error = 67.2746048, gradient norm = 0.0033067 (50 iterations in 10.301s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.274605
[t-SNE] Iteration 300: error = 1.7729300, gradient norm = 0.0011900 (50 iterations in 10.143s)
[t-SNE] Iteration 350: error = 1.3714967, gradient norm = 0.0004818 (50 iterations in 11.128s)
[t-SNE] Iteration 400: error = 1.2036748, gradient norm = 0.0002779 (50 iterations in 10.292s)
[t-SNE] Iteration 450: error = 1.1132656, gradient norm = 0.0001889 (50 iterations in 10.103s)
[t-SNE] Iteration 500: error = 1.0582460, gradient norm = 0.0001434 (50 iterations in 9.733s)
[t-SNE] Iteration 550: error = 1.0222589, gradient norm = 0.0001180 (50 iterations in 9.575s)
[t-SNE] Iteration 600: error = 0.9984865, gradient norm = 0.0001015 (50 iterations in 9.561s)
[t-SNE] Iteration 650: error = 0.9830498, gradient norm = 0.0000958 (50 iterations in 9.624s)
[t-SNE] Iteration 700: error = 0.9726909, gradient norm = 0.0000877 (50 iterations in 10.139s)
[t-SNE] Iteration 750: error = 0.9647216, gradient norm = 0.0000823 (50 iterations in 9.857s)
[t-SNE] Iteration 800: error = 0.9582971, gradient norm = 0.0000755 (50 iterations in 9.929s)
[t-SNE] Iteration 850: error = 0.9531373, gradient norm = 0.0000697 (50 iterations in 9.179s)
[t-SNE] Iteration 900: error = 0.9484153, gradient norm = 0.0000696 (50 iterations in 9.717s)
[t-SNE] Iteration 950: error = 0.9445393, gradient norm = 0.0000659 (50 iterations in 9.614s)
[t-SNE] Iteration 1000: error = 0.9412127, gradient norm = 0.0000674 (50 iterations in 9.508s)
[t-SNE] Error after 1000 iterations: 0.941213
```

In [19]:
```
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 [20]: 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 0.026s...
[t-SNE] Computed neighbors for 5000 samples in 0.583s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
```

```
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.377s
[t-SNE] Iteration 50: error = 80.3592682, gradient norm = 0.0335202 (50 iterations in 24.076s)
[t-SNE] Iteration 100: error = 69.1112671, gradient norm = 0.0036575 (50 iterations in 12.571s)
[t-SNE] Iteration 150: error = 67.6171112, gradient norm = 0.0017708 (50 iterations in 10.925s)
[t-SNE] Iteration 200: error = 67.0565109, gradient norm = 0.0011567 (50 iterations in 10.480s)
[t-SNE] Iteration 250: error = 66.7296524, gradient norm = 0.0009161 (50 iterations in 10.789s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729652
[t-SNE] Iteration 300: error = 1.4983541, gradient norm = 0.0006807 (50 iterations in 13.290s)
[t-SNE] Iteration 350: error = 1.1549147, gradient norm = 0.0001922 (50 iterations in 17.796s)
[t-SNE] Iteration 400: error = 1.0101781, gradient norm = 0.0000912 (50 iterations in 18.827s)
[t-SNE] Iteration 450: error = 0.9388669, gradient norm = 0.0000628 (50 iterations in 17.888s)
[t-SNE] Iteration 500: error = 0.9029322, gradient norm = 0.0000524 (50 iterations in 19.589s)
[t-SNE] Iteration 550: error = 0.8841860, gradient norm = 0.0000482 (50 iterations in 18.979s)
[t-SNE] Iteration 600: error = 0.8722453, gradient norm = 0.0000365 (50 iterations in 15.628s)
[t-SNE] Iteration 650: error = 0.8627461, gradient norm = 0.0000347 (50 iterations in 15.609s)
[t-SNE] Iteration 700: error = 0.8549610, gradient norm = 0.0000312 (50 iterations in 16.253s)
[t-SNE] Iteration 750: error = 0.8487639, gradient norm = 0.0000311 (50 iterations in 16.824s)
[t-SNE] Iteration 800: error = 0.8440317, gradient norm = 0.0000281 (50 iterations in 17.605s)
[t-SNE] Iteration 850: error = 0.8396705, gradient norm = 0.0000250 (50 iterations in 16.034s)
[t-SNE] Iteration 900: error = 0.8354425, gradient norm = 0.0000242 (50 iterations in 15.985s)
[t-SNE] Iteration 950: error = 0.8317489, gradient norm = 0.0000233 (50 iterations in 15.928s)
[t-SNE] Iteration 1000: error = 0.8288577, gradient norm = 0.0000257 (50 iterations in 16.077s)
[t-SNE] Error after 1000 iterations: 0.828858
```

```
In [21]: 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')
```

In [22]:
```
# avoid decoding problems
os.chdir("C:\\Users\\suman\\Downloads\\appliedaidataset\\Quora")
df = pd.read_csv("train.csv")

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

In [23]:
```
df.head()
```

Out[23]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [24]:
```
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 [25]:
```
#This step takes huge time
from scipy.sparse import csr_matrix,coo_matrix
```

```
        from sklearn.cross_validation import train_test_split
        df_all=df
        #take sample
        #_ , df = train_test_split(df, test_size = 5000, random_state=0,stratify = df['is_dup
        # Try TFIDF on texts
        from scipy.sparse import hstack
        tf_idf_vect = TfidfVectorizer(ngram_range=(1,3),min_df = 5,max_features = 50000)
        final_tf_idf2 = tf_idf_vect.fit_transform(df['question2'])
        tf_idf_vect = TfidfVectorizer(ngram_range=(1,3),min_df = 5,max_features = 50000)
        final_tf_idf1 = tf_idf_vect.fit_transform(df['question1'])
        print(final_tf_idf1.shape,final_tf_idf2.shape,df.shape)
        df=df.drop(['id','question1','question2','qid1','qid2','is_duplicate'],axis=1)
        print("last",df.head())
        dense_matrix = np.array(df.as_matrix(columns = None), dtype=float).astype(np.float)
        sparse_matrix = csr_matrix(dense_matrix)
        dfnew=hstack((sparse_matrix, final_tf_idf1,final_tf_idf2)).tocsr()
        print(dfnew.shape)
```

```
(404290, 50000) (404290, 50000) (404290, 6)
last Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]
(404290, 100000)
```

```
In [26]: df=dfnew
         #prepro_features_train.csv (Simple Preprocessing Feartures)
         #nlp_features_train.csv (NLP Features)
         if os.path.isfile('nlp_features_train.csv'):
             dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
         else:
             print("download nlp_features_train.csv from drive or run previous notebook")

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

```
In [27]: 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 = df
         #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)
         #df3_q1 = df3.q1_feats_m
         #df3_q2 = df3.q2_feats_m
```

```
In [28]: # dataframe of nlp features
         df1.head()
```

Out[28]:

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

In [29]:
```
# data before preprocessing
df2.head()
```

Out[29]:

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

In [30]:
```
# Questions 1 tfidf weighted word2vec
#df3_q1.head()
print(df3.shape)
#pd.DataFrame(df3.toarray()).head()
```

```
(404290, 100000)
```

```
In [31]: print("Number of features in nlp dataframe :", df1.shape)
         print("Number of features in preprocessed dataframe :", df2.shape)
         #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 tfidf for Q1 and Q2 :", df3.shape)
         print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df3.shape
```

```
Number of features in nlp dataframe : (404290, 17)
Number of features in preprocessed dataframe : (404290, 12)
Number of features in tfidf for Q1 and Q2 : (404290, 100000)
Number of features in final dataframe  : 100029
```

```
In [32]: # storing the final features to csv file
         from scipy import sparse
         import os
         os.chdir("C:\\Users\\suman\\Downloads\\appliedaidataset\\Quora")

         #if not os.path.isfile('final_features.csv'):
         if not os.path.isfile('dfnew.npz'):
             #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')
             print(df1.head())
             df1=df1.drop(['id'],axis=1)
             #result  = df1.merge(df2, on='id',how='left')
             #instead of tdidf use normal tfidf
             #result  = df1.merge(df2, on='id',how='left')
             print(df1.shape,df1.shape)
             dense_matrix = np.array(df1.as_matrix(columns = None), dtype=float).astype(np.floa
             sparse_matrix = csr_matrix(dense_matrix)
             print("last",df1.head())
             dfnew=hstack((sparse_matrix, df3)).tocsr()
             print("final shape",dfnew.shape)

             #dfnew.to_csv('final_features.csv')
             sparse.save_npz("dfnew.npz", dfnew)
         else:
             your_matrix_back = sparse.load_npz("dfnew.npz")
             print(your_matrix_back.shape)
```

```
(404290, 100026)
```

4. Machine Learning Models

#

4.1 Reading data from file

```
In [33]: from scipy import sparse
         data = sparse.load_npz("dfnew.npz")
         print(data.shape)
         #print(pd.DataFrame(data.toarray()).head())

         #data.drop(data.index[0], inplace=True)
         y_true = pd.read_csv('y.csv',header=None)
         print(y_true.shape)
         #data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace=True)

         # https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
         y_true=y_true.values.ravel()
         print(y_true.shape)
```

```
(404290, 100026)
(404290, 1)
(404290,)
```

4.3 Random train test split(70:30)

```
In [34]: X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test

         # split the data into test and train by maintaining same distribution of output varai
         #X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true
         # split the train data into train and cross validation by maintaining same distributi
         X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, te
```

```
In [35]: print("Number of data points in train data :",X_train.shape)
         print("Number of data points in CV data :",X_cv.shape)
         print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (226402, 100026)
Number of data points in CV data : (56601, 100026)
Number of data points in test data : (121287, 100026)
```

```
In [37]: from collections import Counter
         from scipy.sparse import hstack

         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:  0.6308027314246341 Class 1:  0.36919726857536594
---------- Distribution of output variable in train data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

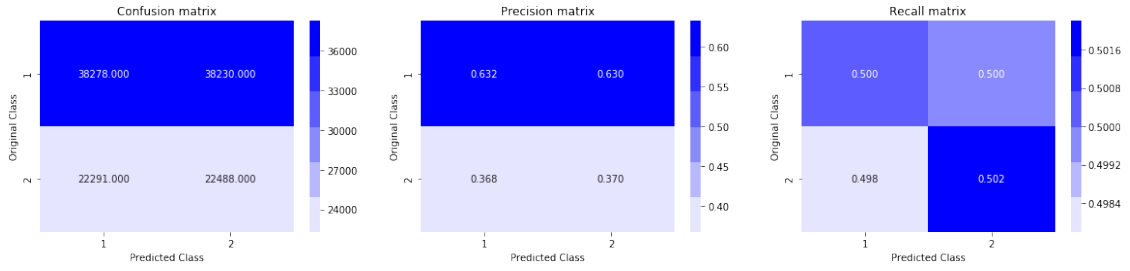
In [38]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

    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=
    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=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 [42]: # 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
         from sklearn.metrics.classification import accuracy_score, log_loss
         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
         loss=log_loss(y_test, predicted_y, eps=1e-15)
         predicted_y =np.argmax(predicted_y, axis=1)
         plot_confusion_matrix(y_test, predicted_y)

         xx='na'
         aa=pd.DataFrame()
         bb=pd.DataFrame({'type':['Random Model'],'hyperparameter':[xx],'log loss CV':['na'], '
                          'log loss Test':[loss]})
         aa=aa.append(bb)
```

```
Log loss on Test Data using Random Model 0.8831717718040545
```

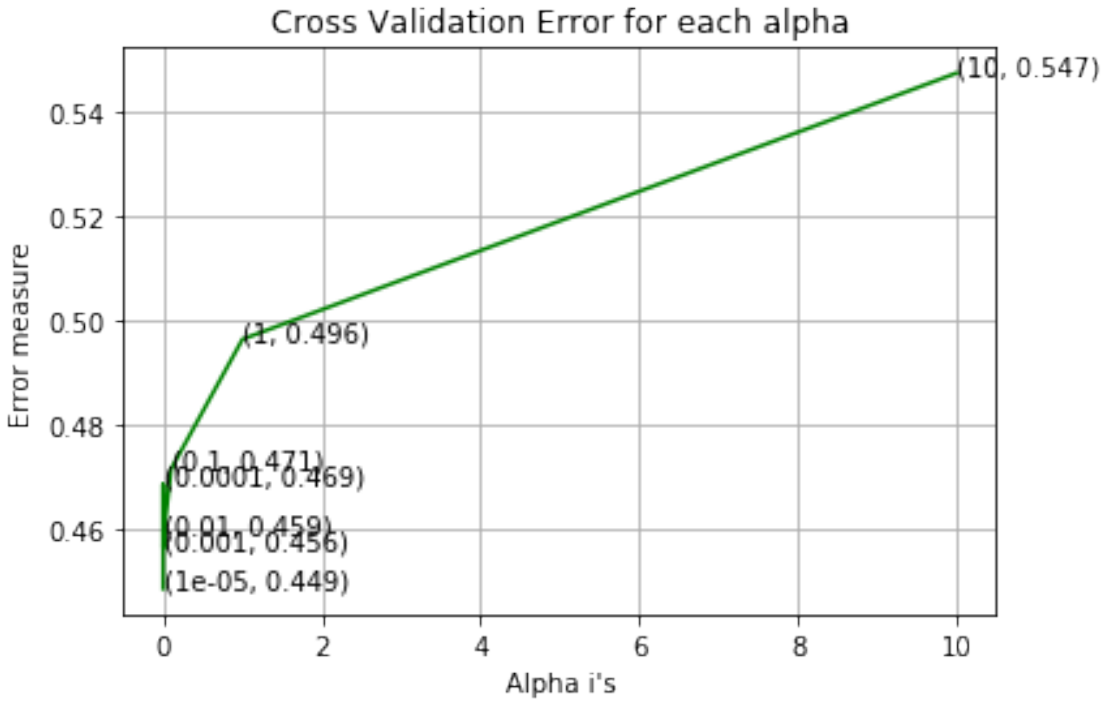


4.4 Logistic Regression with hyperparameter tuning

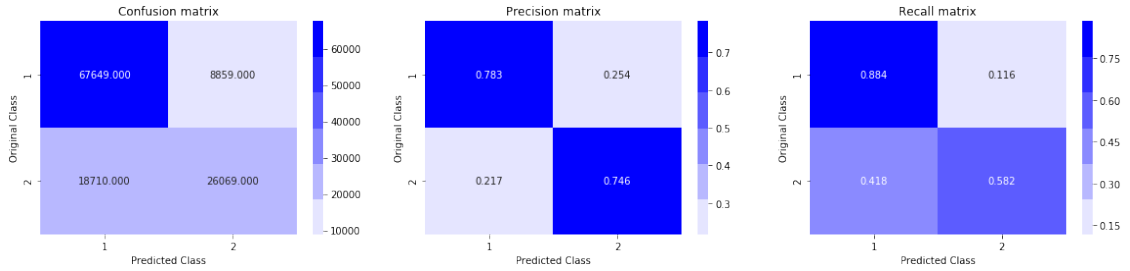
In [43]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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


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

xx='alpha='+str(alpha[best_alpha])
bb=pd.DataFrame({'type':['Logistic'],'hyperparameter':[xx],'log loss CV':[log_loss(y_c
                 'log loss Test':[log_loss(y_test, sig_clf.predict_proba(X_test))]})
```

```
        aa=aa.append(bb)


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

```
For values of alpha =  1e-05 The log loss is: 0.4485498246568632
For values of alpha =  0.0001 The log loss is: 0.46859055984500175
For values of alpha =  0.001 The log loss is: 0.45637448598068986
For values of alpha =  0.01 The log loss is: 0.45913913696692205
For values of alpha =  0.1 The log loss is: 0.47144540753938097
For values of alpha =  1 The log loss is: 0.49628369529528527
For values of alpha =  10 The log loss is: 0.547245977516822
```




```
For values of best alpha =  1e-05 The train log loss is: 0.446711642422553
For values of best alpha =  1e-05 The test log loss is: 0.4485498246568632
Total number of data points : 121287
```

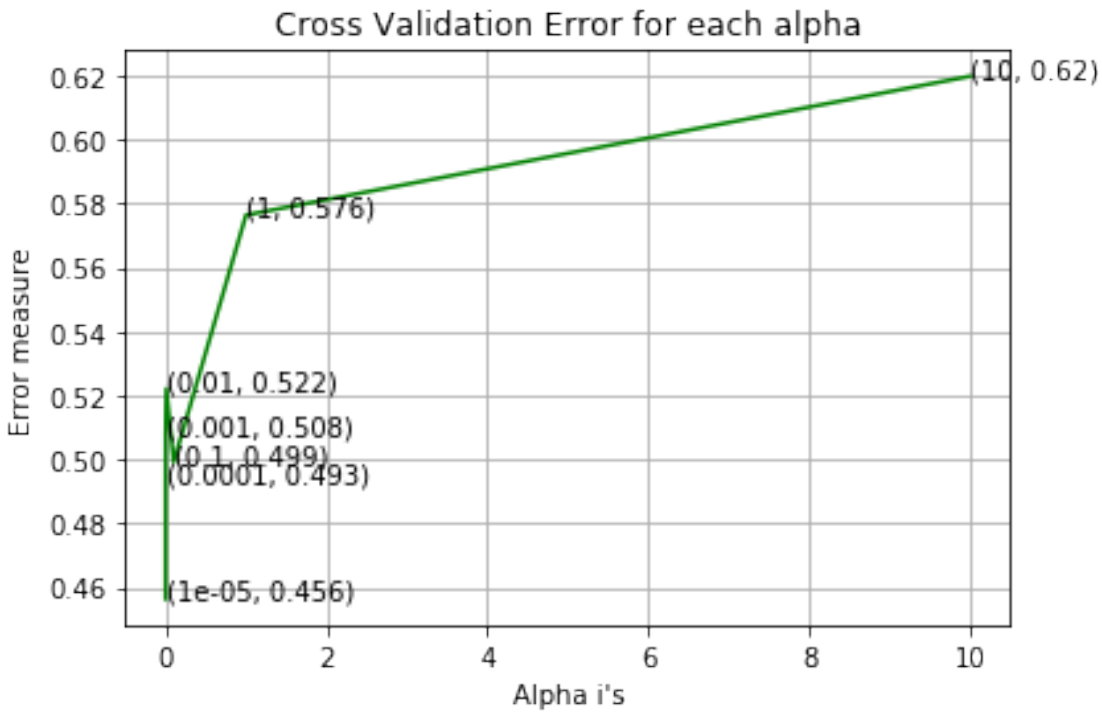


4.5 Linear SVM with hyperparameter tuning

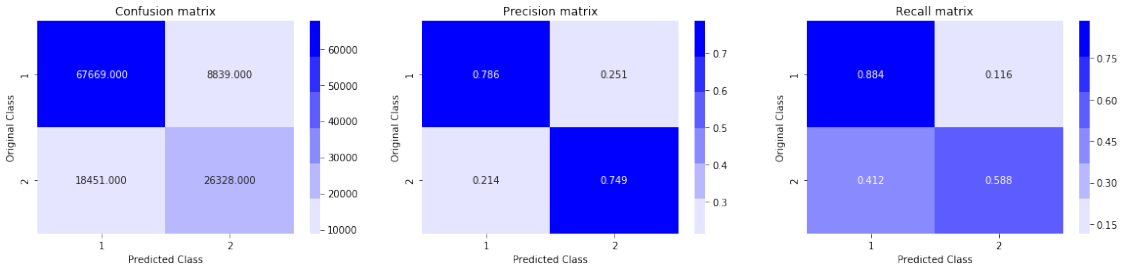
```
In [44]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

         # 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)

        xx='alpha='+str(alpha[best_alpha])
        bb=pd.DataFrame({'type':['Linear SVM'],'hyperparameter':[xx],'log loss CV':[log_loss(y
                         'log loss Test':[log_loss(y_test, sig_clf.predict_proba(X_test))]})
        aa=aa.append(bb)


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

```
For values of alpha =  1e-05 The log loss is: 0.456437670191809
For values of alpha =  0.0001 The log loss is: 0.4930466773334815
For values of alpha =  0.001 The log loss is: 0.5080321321063807
For values of alpha =  0.01 The log loss is: 0.5221033142438856
For values of alpha =  0.1 The log loss is: 0.4991434795905807
For values of alpha =  1 The log loss is: 0.5763297198165512
For values of alpha =  10 The log loss is: 0.6197004557270542
```




```
For values of best alpha =  1e-05 The train log loss is: 0.45490040115182334
For values of best alpha =  1e-05 The test log loss is: 0.456437670191809
Total number of data points : 121287
```



4.6 XGBoost

```
In [49]: #from xgboost import XGBClassifier
         import warnings
         warnings.filterwarnings("ignore")
         #installing xgboost was difficult, first install py-xgboost, then getting probelm for
         #ages and installed : pip installed xgboost-0.80-cp35-cp35m-win_amd64  : for python 3
         from xgboost import XGBClassifier
         #hyperparameter tunning
```

```
        tuned_parameters={'learning_rate':[.1,.1],'n_estimators':[10,20,40],'max_depth':[6,8]}
        model = RandomizedSearchCV(XGBClassifier(), tuned_parameters, random_state=1, scoring
        print(X_train.shape,y_train.shape,type(X_train),type(y_train))
        model.fit(X_train, y_train)
```

```
(226402, 100026) (226402,) <class 'scipy.sparse.csr.csr_matrix'> <class 'numpy.ndarray'>
```

```
In [47]: xx='learning_rate='+str(model.best_estimator_.learning_rate)+'n_estimator='+str(model
         #execute till this smuk
         model=XGBClassifier(learning_rate=model.best_estimator_.learning_rate,n_estimators=mo
         model.fit(X_train, y_train)
         sig_clf = CalibratedClassifierCV(model, method="sigmoid")
         sig_clf.fit(X_train, y_train)
         predict_y = sig_clf.predict_proba(X_test)
         print('log loss',log_loss(y_test, sig_clf.predict_proba(X_test)))

         bb=pd.DataFrame({'type':['xgboost '],'hyperparameter':[xx],'log loss CV':[log_loss(y_
                          'log loss Test':[log_loss(y_test, sig_clf.predict_proba(X_test))]})
         aa=aa.append(bb)
```

```
log loss 0.35992486111830874
```

```
In [48]: aa
```

Out[48]:

	hyperparameter	log loss CV	log loss Test	type
0	na	na	0.883172	Random Model
0	alpha=1e-05	0.446188	0.448550	Logistic
0	alpha=1e-05	0.455562	0.456438	Linear SVM
0	learning_rate=0.1n_estimator=40max_depth=6	0.360693	0.359925	xgboost

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.