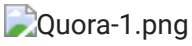


Quora Question Pairs

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 pe 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 cause seekers to spend more time finding the best answer to their question, and make writers feel they need to ans Quora values canonical questions because they provide a better experience to active seekers and writers, and offer 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/AACZdtsAp
- 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-4c1cf93f

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
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

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

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

```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /gdrive
/gdrive
```

3. Exploratory Data Analysis

```
pip install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437aefa0d0a92ccfb
     |████████████████████████████████| 184kB 2.8MB/s
Building wheels for collected packages: distance
  Building wheel for distance (setup.py) ... done
  Created wheel for distance: filename=Distance-0.1.3-cp36-none-any.whl size=16261 sha25
  Stored in directory: /root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d0f12db1c66dbae
Successfully built distance
Installing collected packages: distance
Successfully installed distance-0.1.3
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.6/dist-packages (0.90)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from xgb
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from xgb
```

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

import re
```

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

3.1 Reading data and basic stats

```
df1 = pd.read_csv("./My Drive/Quora/train.csv")

print("Number of data points:",df1.shape[0])
```

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

```
df1.head(2)
```

	id	qid1	qid2	question1	
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian gc

```
df = df1[df1['is_duplicate'] == 0].iloc[:25000,:]
```

```
df.shape
```

```
(25000, 6)
```

```
df = df.append(df1[df1['is_duplicate'] == 1].iloc[:25000,:], ignore_index = True)
```

```
df['is_duplicate'].value_counts()
```

```
1    25000
0    25000
Name: is_duplicate, dtype: int64
```

```
df.shape
```

```
(50000, 6)
```

```
df = df.dropna(how='any',axis=0)
df.shape
```

```
(50000, 6)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 6 columns):
id              50000 non-null int64
qid1            50000 non-null int64
qid2            50000 non-null int64
question1       50000 non-null object
question2       50000 non-null object
is_duplicate    50000 non-null int64
dtypes: int64(4), object(2)
memory usage: 2.7+ MB
```

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

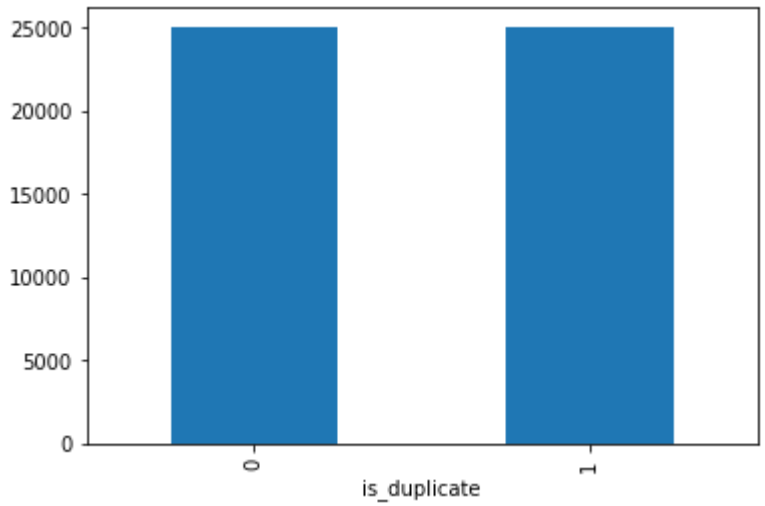
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7facb87ca940>
```




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

```
~> Total number of question pairs for training:
   50000
```

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

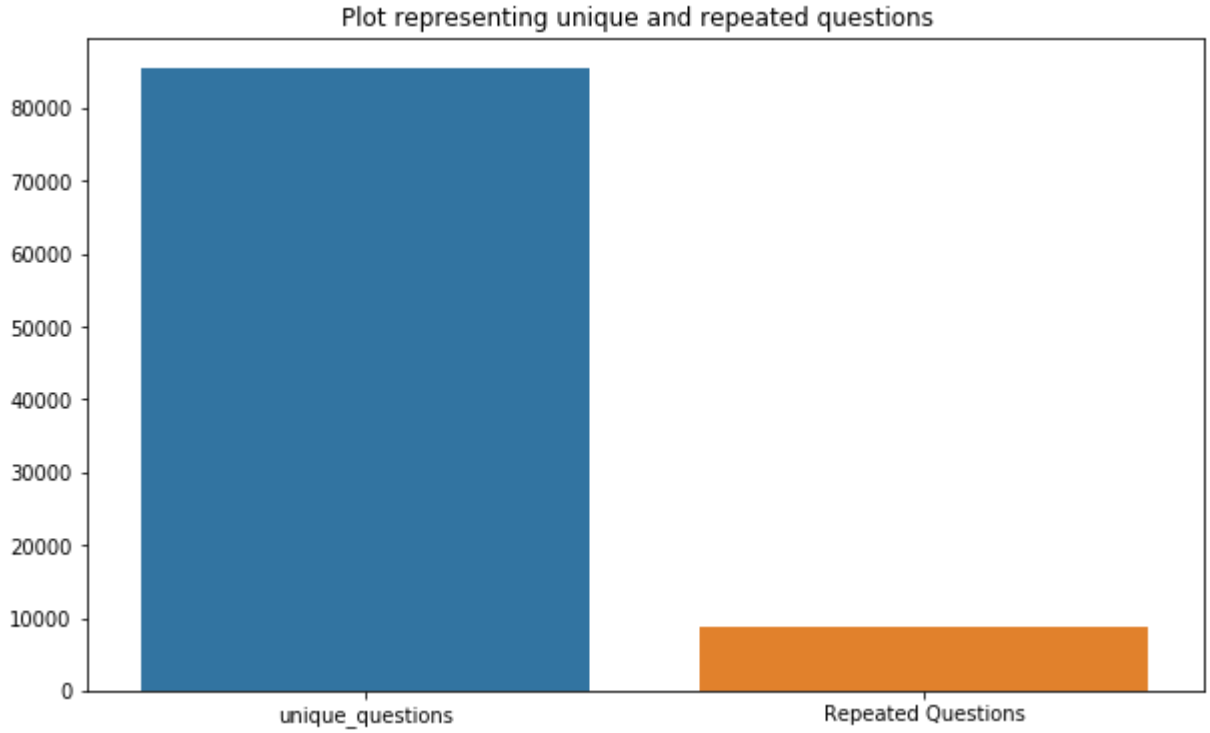
```
~> Question pairs are not Similar (is_duplicate = 0):
   50.0%

~> Question pairs are Similar (is_duplicate = 1):
   50.0%
```

3.2.2 Number of unique questions

```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

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

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 85408

Number of unique questions that appear more than one time: 8724 (10.214499812663918%)

Max number of times a single question is repeated: 16
```

```
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

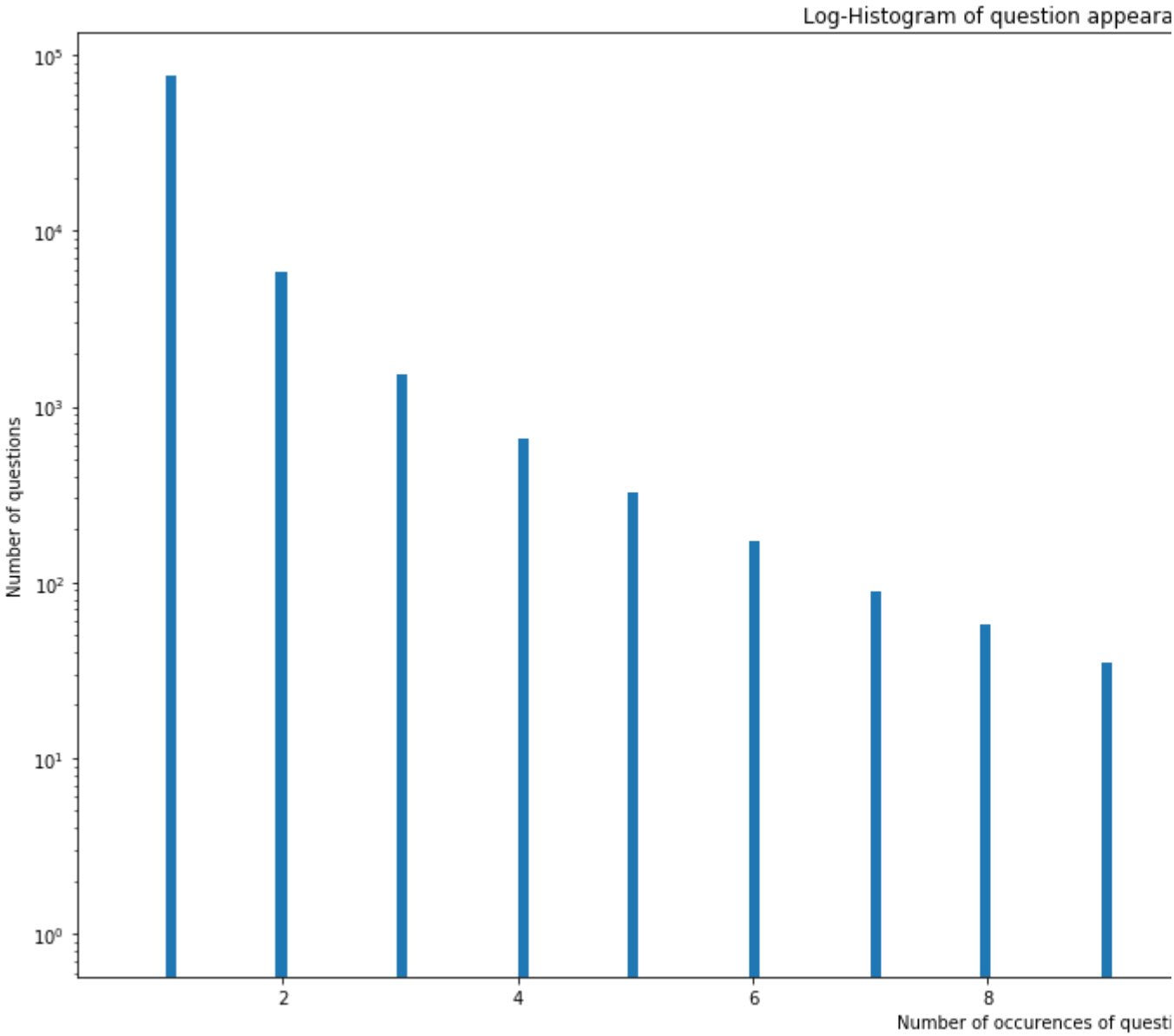
```
#checking whether there are any repeated pair of questions
pair_duplicates = df[['qid1','qid2','is_duplicate']].groupby(['qid1','qid2']).count().reset_index()
print ("Number of duplicate questions",(pair_duplicates).shape[0] - df.shape[0])
```

```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

```
plt.figure(figsize=(20, 10))
plt.hist(qids.value_counts(), bins=160)
plt.yscale('log', nonposy='clip')
plt.title('Log-Histogram of question appearance counts')
plt.xlabel('Number of occurences of question')
plt.ylabel('Number of questions')
print ('Maximum number of times a single question is repeated: {}\n'.format(max(qids.value_counts())
```

```
Maximum number of times a single question is repeated: 16
```



3.2.5 Checking for NULL values

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

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

- There are two rows with null values in question2

```
# 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

```
if os.path.isfile('./My Drive/Quora2/df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("./My Drive/Quora2/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("./My Drive/Quora2/df_fe_without_preprocessing_train.csv", index=False)

df.head(2)
```

	id	qid1	qid2	question1	
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian gc

```
df.shape
```

```
(50000, 17)
```

3.3.1 Analysis of some of the extracted features

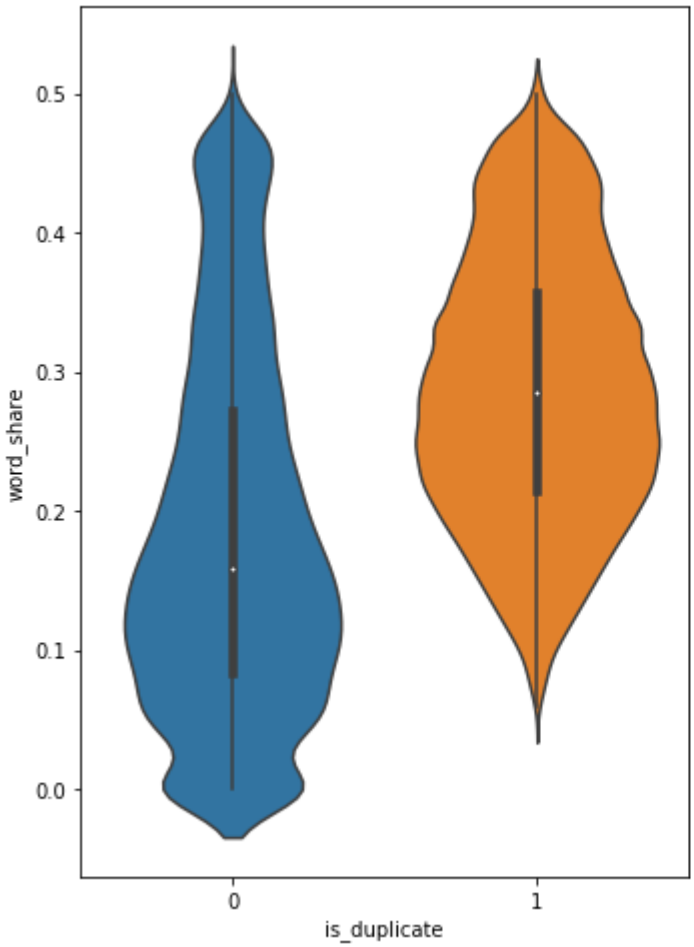
- Here are some questions have only one single words.

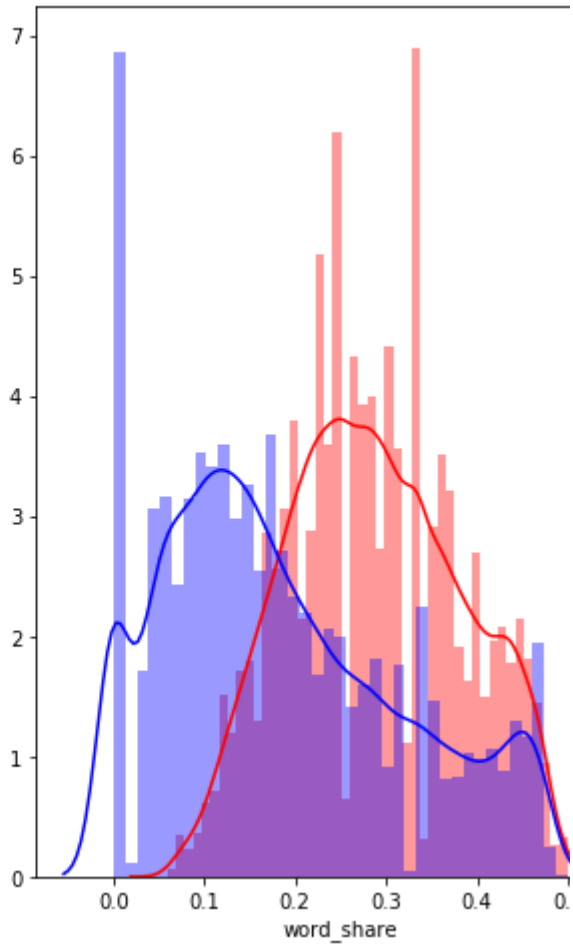
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

print ("Minimum length of the questions in question2 : " , min(df['q2_n_words']))

print ("Number of Questions with minimum length [question1] :", df[df['q1_n_words']== 1].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']== 1].shape[0])
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 9
Number of Questions with minimum length [question2] : 1
```

3.3.1.1 Feature: word_share

```
plt.figure(figsize=(12, 8))

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

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

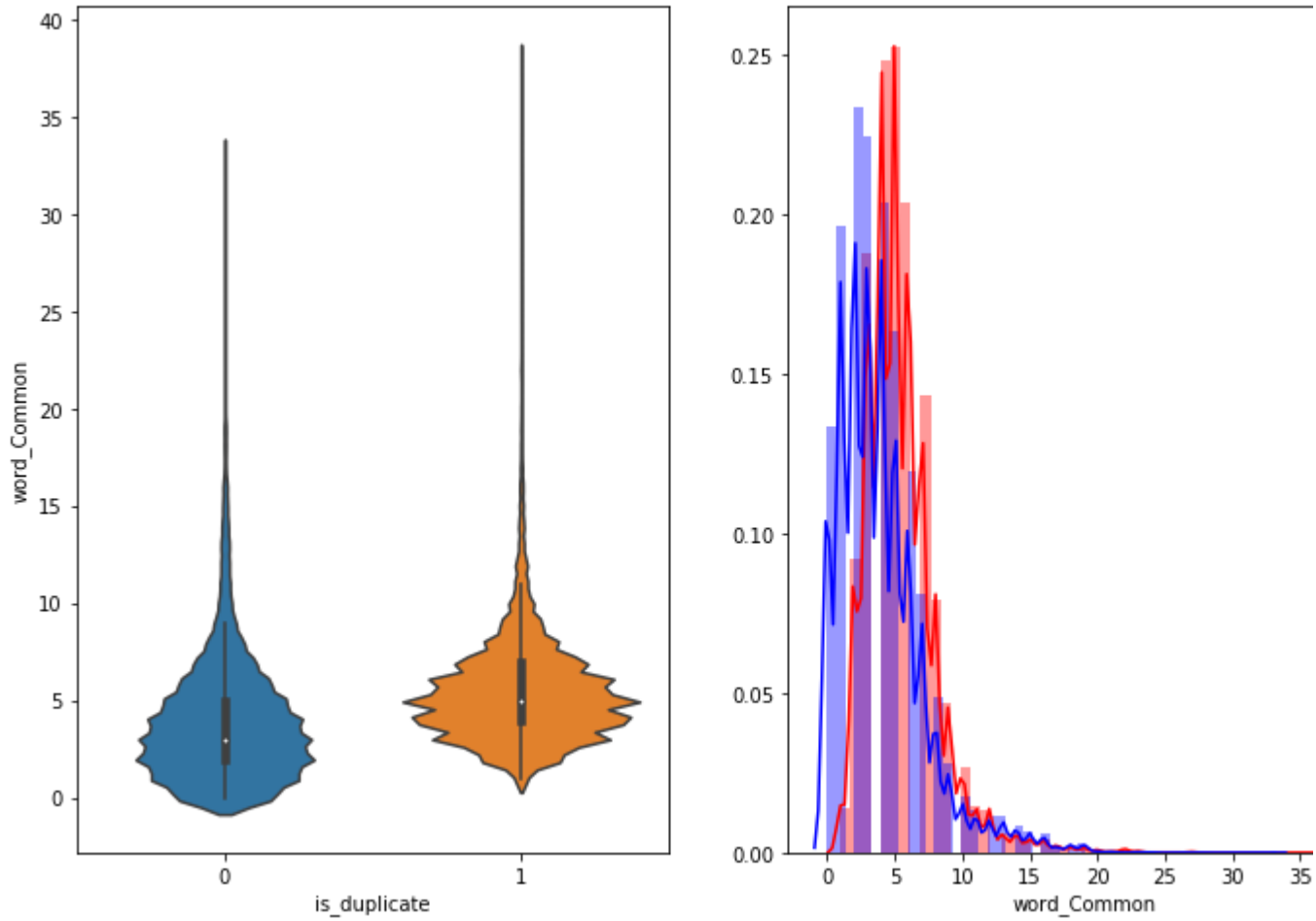




- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

```
plt.figure(figsize=(12, 8))

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

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



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

1.2.1 : EDA: Advanced Feature Extraction.

```
pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.6/dist-packages (0.1
```

```
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
```

3.4 Preprocessing of Text

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

```
# To get the results in 4 decemal points
import nltk
nltk.download('stopwords')
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("’", "'").replace("’", "'")\
                           .replace("won't", "will not").replace("cannot", "can not").replace("can't
                           .replace("n't", " not").replace("what's", "what is").replace("it's", "it
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").replace("'s", " own"
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


    porter = PorterStemmer()
    pattern = re.compile('\W')

    if type(x) == type(''):
        x = re.sub(pattern, ' ', x)


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

```
        x = example1.get_text()

    return x
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

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 Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))

- **ctc_min** : Ratio of common_token_count to min lenghth of token count of Q1 and Q2
  ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))

- **ctc_max** : Ratio of common_token_count to max lenghth of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

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

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

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

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

- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzz

- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuz

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzz

- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy

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

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

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

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

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

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

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

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


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

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

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

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

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

# get the Longest Common sub string

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

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

    print("token features...")

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

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

```
if os.path.isfile('./My Drive/Quora2/nlp_features_train.csv'):
    df = pd.read_csv("./My Drive/Quora2/nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = extract_features(df)
    df.to_csv("./My Drive/Quora2/nlp_features_train.csv", index=False)
df.head(2)
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2le
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	5
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	1	1	51	8

```
df.shape
```

```
(50000, 32)
```

3.5.1 Analysis of extracted features

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

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

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

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

#Saving the np array into a text file
np.savetxt('./My Drive/Quora2/train_p.txt', p, delimiter=' ', fmt='%s')
np.savetxt('./My Drive/Quora2/train_n.txt', n, delimiter=' ', fmt='%s')
```

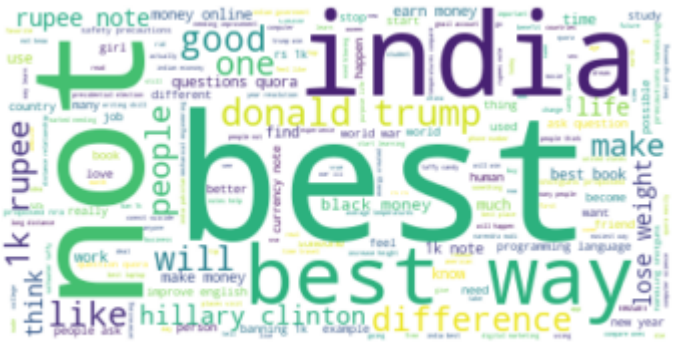
```
Number of data points in class 1 (duplicate pairs) : 50000
Number of data points in class 0 (non duplicate pairs) : 50000
```

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

textp_w = open(path.join(d, './My Drive/Quora2/train_p.txt')).read()
textn_w = open(path.join(d, './My Drive/Quora2/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 : 2695444
Total number of words in non duplicate pair questions : 3254369
```

```
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()
```

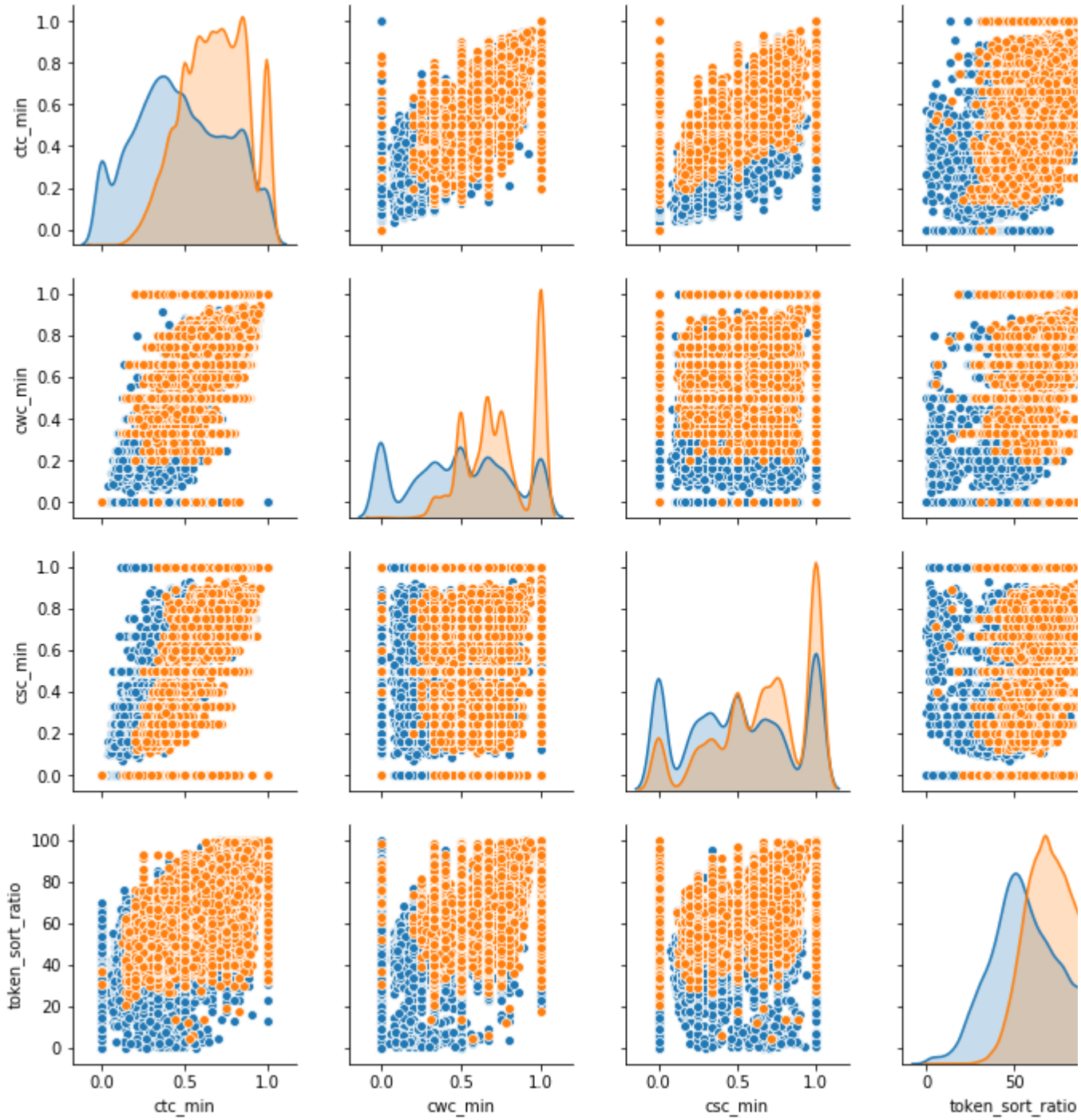
```
Word Cloud for Duplicate Question pairs
```



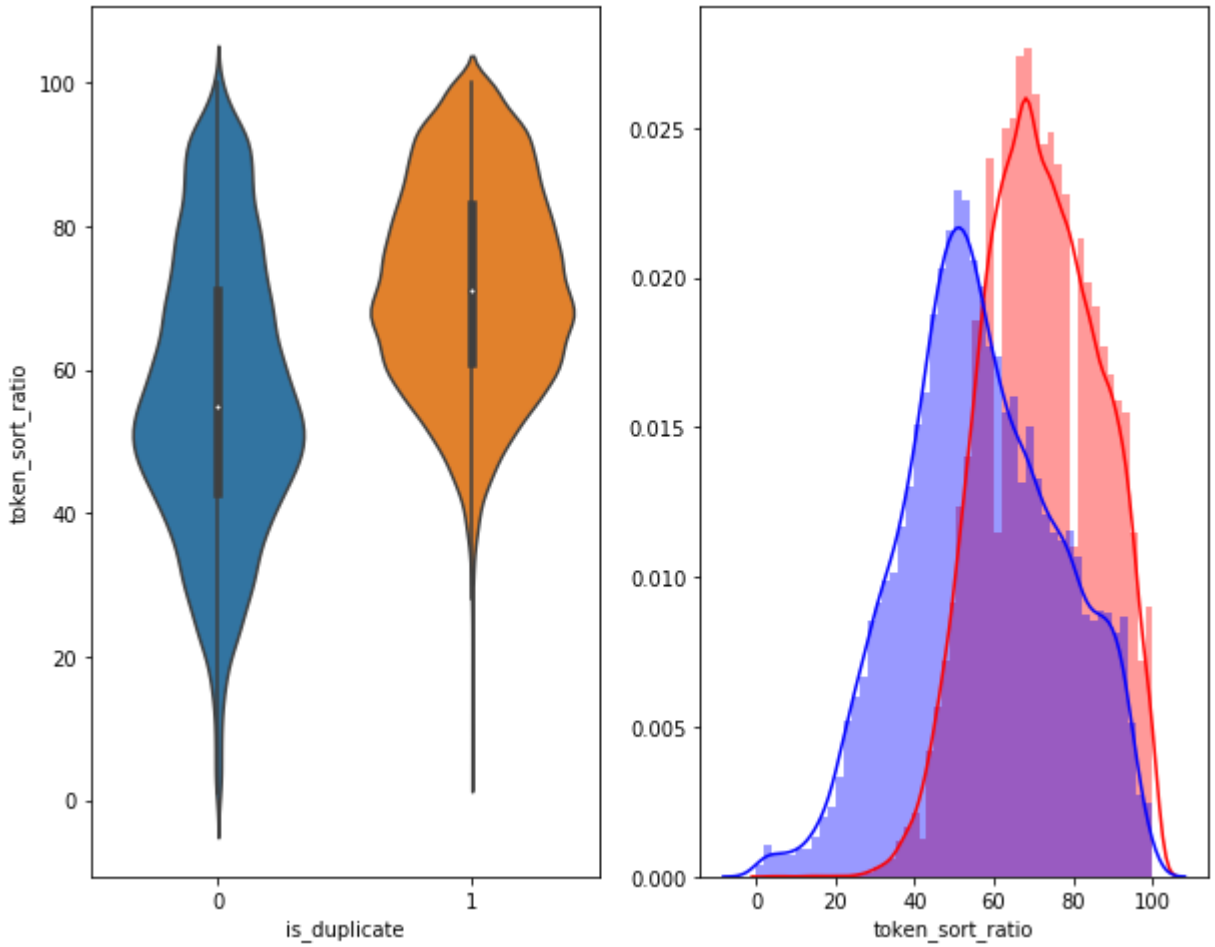

```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word Cloud for non-Duplicate Question pairs:

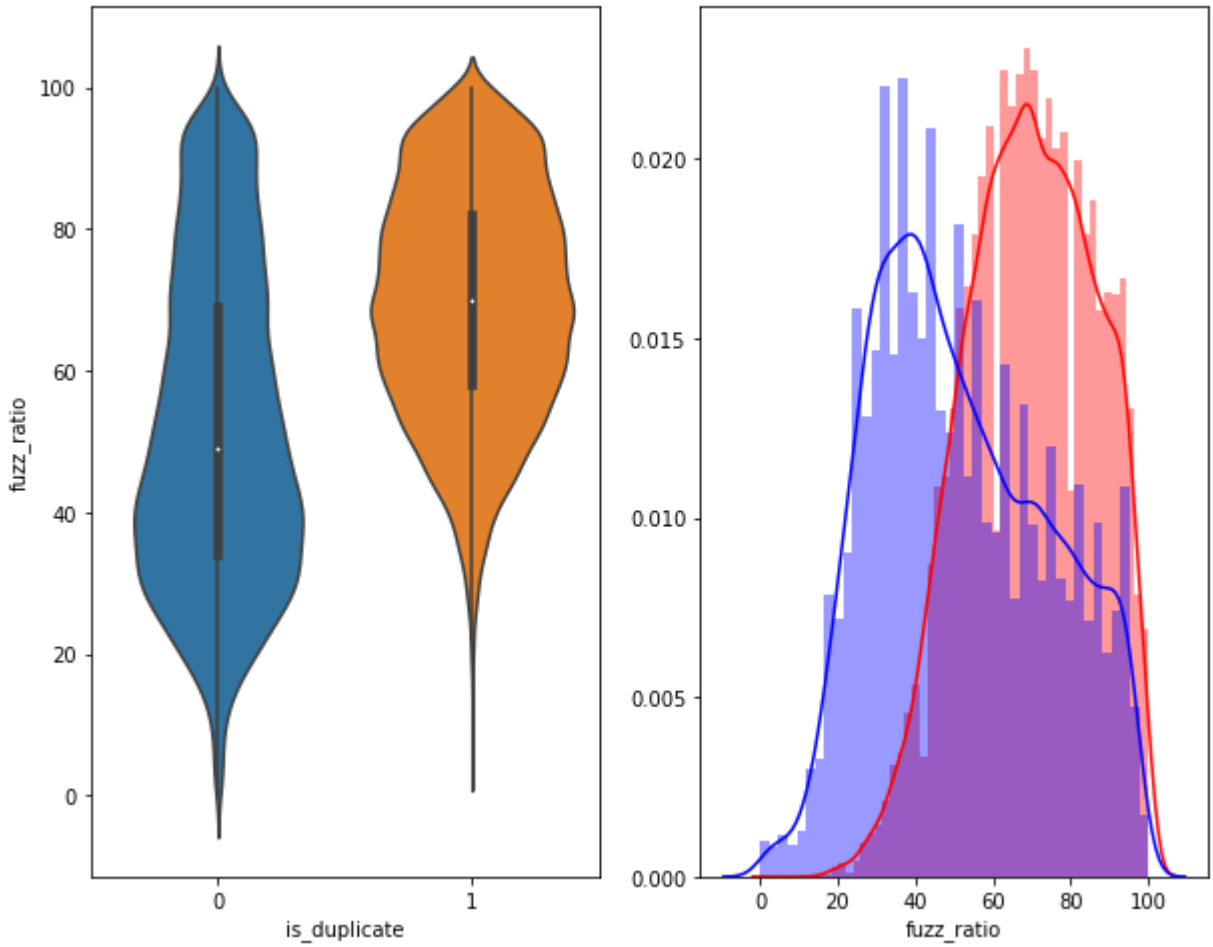

```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is
plt.show()
```




```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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

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




```
plt.figure(figsize=(10, 8))

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

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



3.5.2 Visualization

```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 3 di

from sklearn.preprocessing import MinMaxScaler

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

```
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 0.021s...
[t-SNE] Computed neighbors for 5000 samples in 0.421s...
[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: 0.113036
[t-SNE] Computed conditional probabilities in 0.288s
[t-SNE] Iteration 50: error = 82.2266464, gradient norm = 0.0456455 (50 iterations in 3.
[t-SNE] Iteration 100: error = 70.4080124, gradient norm = 0.0094598 (50 iterations in 2
[t-SNE] Iteration 150: error = 68.5708313, gradient norm = 0.0053066 (50 iterations in 1
[t-SNE] Iteration 200: error = 67.7776642, gradient norm = 0.0037753 (50 iterations in 1
[t-SNE] Iteration 250: error = 67.3147583, gradient norm = 0.0031309 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.314758
[t-SNE] Iteration 300: error = 1.8008114, gradient norm = 0.0011846 (50 iterations in 1.
[t-SNE] Iteration 350: error = 1.4070239, gradient norm = 0.0005074 (50 iterations in 1.
[t-SNE] Iteration 400: error = 1.2410451, gradient norm = 0.0002899 (50 iterations in 1.
[t-SNE] Iteration 450: error = 1.1534489, gradient norm = 0.0001925 (50 iterations in 1.
[t-SNE] Iteration 500: error = 1.0993203, gradient norm = 0.0001432 (50 iterations in 1.
[t-SNE] Iteration 550: error = 1.0641595, gradient norm = 0.0001158 (50 iterations in 1.
[t-SNE] Iteration 600: error = 1.0409743, gradient norm = 0.0001016 (50 iterations in 1.
[t-SNE] Iteration 650: error = 1.0250087, gradient norm = 0.0000896 (50 iterations in 1.
[t-SNE] Iteration 700: error = 1.0134697, gradient norm = 0.0000812 (50 iterations in 1.
[t-SNE] Iteration 750: error = 1.0046519, gradient norm = 0.0000740 (50 iterations in 1.
[t-SNE] Iteration 800: error = 0.9977633, gradient norm = 0.0000698 (50 iterations in 1.
[t-SNE] Iteration 850: error = 0.9919611, gradient norm = 0.0000667 (50 iterations in 1.
[t-SNE] Iteration 900: error = 0.9870079, gradient norm = 0.0000615 (50 iterations in 1.
[t-SNE] Iteration 950: error = 0.9827231, gradient norm = 0.0000587 (50 iterations in 1.
[t-SNE] Iteration 1000: error = 0.9791667, gradient norm = 0.0000586 (50 iterations in 1
[t-SNE] KL divergence after 1000 iterations: 0.979167
```

```
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 0.014s...
[t-SNE] Computed neighbors for 5000 samples in 0.430s...
[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: 0.113036
[t-SNE] Computed conditional probabilities in 0.290s
[t-SNE] Iteration 50: error = 83.4003448, gradient norm = 0.0391808 (50 iterations in 10
[t-SNE] Iteration 100: error = 69.1974030, gradient norm = 0.0034733 (50 iterations in 5
[t-SNE] Iteration 150: error = 67.9046936, gradient norm = 0.0018418 (50 iterations in 4
[t-SNE] Iteration 200: error = 67.2827225, gradient norm = 0.0011958 (50 iterations in 5
[t-SNE] Iteration 250: error = 66.9548721, gradient norm = 0.0009378 (50 iterations in 5
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.954872
[t-SNE] Iteration 300: error = 1.5419589, gradient norm = 0.0007568 (50 iterations in 7.
[t-SNE] Iteration 350: error = 1.2043211, gradient norm = 0.0002053 (50 iterations in 9.
[t-SNE] Iteration 400: error = 1.0542364, gradient norm = 0.0000942 (50 iterations in 9.
[t-SNE] Iteration 450: error = 0.9784088, gradient norm = 0.0000613 (50 iterations in 9.
[t-SNE] Iteration 500: error = 0.9383177, gradient norm = 0.0000487 (50 iterations in 9.
[t-SNE] Iteration 550: error = 0.9166503, gradient norm = 0.0000413 (50 iterations in 9.
[t-SNE] Iteration 600: error = 0.9023421, gradient norm = 0.0000386 (50 iterations in 9.
[t-SNE] Iteration 650: error = 0.8934093, gradient norm = 0.0000342 (50 iterations in 9.
[t-SNE] Iteration 700: error = 0.8875222, gradient norm = 0.0000299 (50 iterations in 9.
[t-SNE] Iteration 750: error = 0.8813444, gradient norm = 0.0000283 (50 iterations in 9.
[t-SNE] Iteration 800: error = 0.8762089, gradient norm = 0.0000276 (50 iterations in 9.
[t-SNE] Iteration 850: error = 0.8719776, gradient norm = 0.0000253 (50 iterations in 9.
[t-SNE] Iteration 900: error = 0.8678088, gradient norm = 0.0000233 (50 iterations in 9.
[t-SNE] Iteration 950: error = 0.8639071, gradient norm = 0.0000234 (50 iterations in 9.
[t-SNE] Iteration 1000: error = 0.8606573, gradient norm = 0.0000215 (50 iterations in 9
[t-SNE] KL divergence after 1000 iterations: 0.860657
```

```
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

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

```
df.dropna(how='any', inplace = True)
```

```
df.shape
```

```
(49996, 32)
```

```
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_))
```

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 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)
```

```
100%|██████████| 49996/49996 [07:42<00:00, 107.98it/s]
```

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question2'])):
    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['q2_feats_m'] = list(vecs1)
```

```
100%|██████████| 49996/49996 [07:42<00:00, 108.00it/s]
```

```
df.head(2)
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2le
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	5
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	1	1	51	8

```
columns = ['freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'w

from sklearn.preprocessing import StandardScaler
for column in columns:
  scalar = StandardScaler()
  scalar.fit(df[column].values.reshape(-1,1))
  print(column + ": ")
  print(f"Mean : {scalar.mean_[0]}, Standard deviation : {np.sqrt(scalar.var_[0])}")
  standardized_df = scalar.transform(df[column].values.reshape(-1, 1))
  df[column] = pd.DataFrame(standardized_df)
```

```
freq_qid1:
Mean : 1.3507880630450435, Standard deviation : 0.9993985605377591
freq_qid2:
Mean : 1.3659492759420753, Standard deviation : 1.050095260893902
q1len:
Mean : 58.11368909512761, Standard deviation : 28.955992674826547
q2len:
Mean : 58.59250740059205, Standard deviation : 32.400150688640586
q1_n_words:
Mean : 10.710156812545003, Standard deviation : 5.2350864498900895
q2_n_words:
Mean : 10.899551964157133, Standard deviation : 6.024640628523511
word_Common:
Mean : 4.697795823665893, Standard deviation : 3.044996174403431
word_Total:
Mean : 20.681774541963357, Standard deviation : 8.31996865435955
word_share:
Mean : 0.23655357186510334, Standard deviation : 0.12511841464427614
freq_q1+q2:
Mean : 2.716737338987119, Standard deviation : 1.7427253516337018
freq_q1-q2:
Mean : 0.41135290823265863, Standard deviation : 0.9984598517475738
cwc_min:
Mean : 0.6202869061117658, Standard deviation : 0.29799512252192734
cwc_max:
Mean : 0.48850342946711584, Standard deviation : 0.2687320176073759
csc_min:
Mean : 0.5905576611181693, Standard deviation : 0.3329424011819995
csc_max:
Mean : 0.4522941271853022, Standard deviation : 0.29749451838170277
ctc_min:
Mean : 0.5813714725096985, Standard deviation : 0.2576382564181938
ctc_max:
Mean : 0.45613208924668663, Standard deviation : 0.24083611387488318
last_word_eq:
Mean : 0.3566085286822946, Standard deviation : 0.4789977932656304
first_word_eq:
Mean : 0.5396631730538443, Standard deviation : 0.4984243500304743
abs_len_diff:
Mean : 3.5104808384670774, Standard deviation : 4.674028014875774
mean_len:
Mean : 10.98501880150412, Standard deviation : 5.0096031051879
token_set_ratio:
Mean : 73.79524361948955, Standard deviation : 19.756939163500583
token_sort_ratio:
Mean : 64.15737258980718, Standard deviation : 18.62926514414492
fuzz_ratio:
Mean : 60.65935274821986, Standard deviation : 21.43542364694602
fuzz_partial_ratio:
Mean : 66.50538043043443, Standard deviation : 18.300697918298855
longest_substr_ratio:
Mean : 0.4054397665673164, Standard deviation : 0.2275867841543115
```

```
df.head(2)
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	
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	-0.350999	-0.348492	0.272355	-
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	-0.350999	-0.348492	-0.245672	

```
df_q1 = pd.DataFrame(df.q1_feats_m.values.tolist(), index= df.index)
df_q2 = pd.DataFrame(df.q2_feats_m.values.tolist(), index= df.index)
```

```
df_q1.head(2)
```

	0	1	2	3	4	5	6	
0	183.066797	-137.327993	-56.972606	-153.662772	-88.786487	4.487015	135.681654	46.1
1	80.836472	-126.260577	-11.482833	-59.296239	-113.777659	-45.185599	13.408010	20.7

2 rows × 96 columns

```
df_q1['id']=df['id']
df_q2['id']=df['id']
df1  = df.merge(df_q1, on='id',how='left')
```

```
df1.head(2)
```

	id	qid1	qid2	question1	question2	freq_qid1	freq_qid2	q1len	q2len	q1_n
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...	-0.350999	-0.348492	0.272355	-0.049151	0.(
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	-0.350999	-0.348492	-0.245672	0.907634	-0.5

2 rows × 225 columns

```
df1  = df1.merge(df_q2, on='id',how='left')
```

```
df1.head(2)
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	
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	-0.350999	-0.348492	0.272355	-
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	-0.350999	-0.348492	-0.245672	

2 rows × 226 columns

```
df1.shape
```

```
(49996, 226)
```

4.3 Random train test split(70:30)

```
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 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



df1.dropna(how='any', inplace = True)



y = df1["is_duplicate"]
df1.drop(["is_duplicate"], axis = 1, inplace = True)



df1.shape
```

```
(49992, 226)
```

```
from sklearn.model_selection import train_test_split

X_train,X_test, y_train, y_test = train_test_split(df, y, stratify=y, test_size=0.3)



X_train.shape
```

```
(34994, 225)
```

```
print(" "*10, "Distribution of output variable in train data", " "*10)
```

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

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

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

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamension
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamension
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

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

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

```
    plt.show()
```

TFIDF Features

```
from sklearn.feature_extraction.text import TfidfVectorizer
print("_"*25+"Question1 TFIDF"+"_"*25)
vectorizer8 = TfidfVectorizer(min_df=10)
vectorizer8.fit(X_train['question1'])
q1_tfidf_train = vectorizer8.transform(X_train['question1'])
q1_tfidf_test = vectorizer8.transform(X_test['question1'])
print("Shape of train matrix after one hot encodig ",q1_tfidf_train.shape)
print("Shape of test matrix after one hot encodig ",q1_tfidf_test.shape)
print("="*100)


print("_"*25+"Question2 TFIDF"+"_"*25)
vectorizer9 = TfidfVectorizer(min_df=10)
vectorizer9.fit_transform(X_train['question2'])
q2_tfidf_train = vectorizer9.transform(X_train['question2'])
q2_tfidf_test = vectorizer9.transform(X_test['question2'])
print("Shape of train matrix after one hot encodig ",q2_tfidf_train.shape)
print("Shape of test matrix after one hot encodig ",q2_tfidf_test.shape)
```

```
_________________________Question1 TFIDF_________________________
Shape of train matrix after one hot encodig  (34994, 3092)
Shape of test matrix after one hot encodig  (14998, 3092)
====================================================================================================
_________________________Question2 TFIDF_________________________
Shape of train matrix after one hot encodig  (34994, 3140)
Shape of test matrix after one hot encodig  (14998, 3140)
```

```
X_train.drop(['qid1', 'qid2', 'question1', 'question2','id', 'q1_feats_m','q2_feats_m'], axis=1, inp
X_test.drop(['qid1', 'qid2', 'question1', 'question2','id', 'q1_feats_m','q2_feats_m'], axis=1, inpl
```

```
X_train.head(2)
```

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wc
15464	-0.350999	-0.348492	-0.591024	1.771828	-0.899729	1.012583	-0.885977	
30161	-0.350999	-0.348492	-0.245672	-0.080015	-0.326672	-0.149312	0.756061	

2 rows × 218 columns

```
X_train.shape
```

```
(34994, 218)
```

```
X_test.shape
```

```
(14998, 218)
```

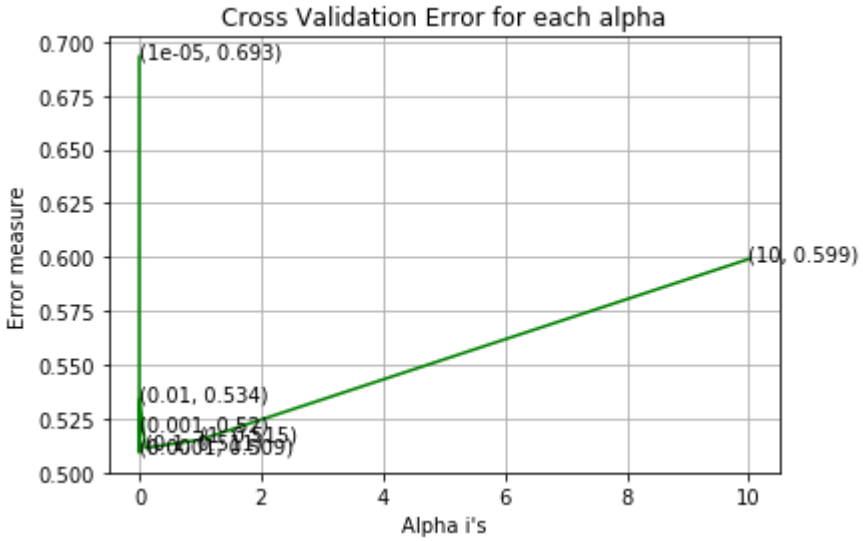
```
from sklearn import preprocessing

for f in X_train.columns:
    if X_train[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(X_train[f].values))
        X_train[f] = lbl.transform(list(X_train[f].values))

for f in X_test.columns:
    if X_test[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(X_test[f].values))
        X_test[f] = lbl.transform(list(X_test[f].values))

X_train.fillna((-999), inplace=True)
X_test.fillna((-999), inplace=True)

train1=np.array(X_train)
test1=np.array(X_test)
X_train = train1.astype(float)
X_test = test1.astype(float)
```

Logistic Regression with hyperparameter tuning (TFIDF weighted W2V)

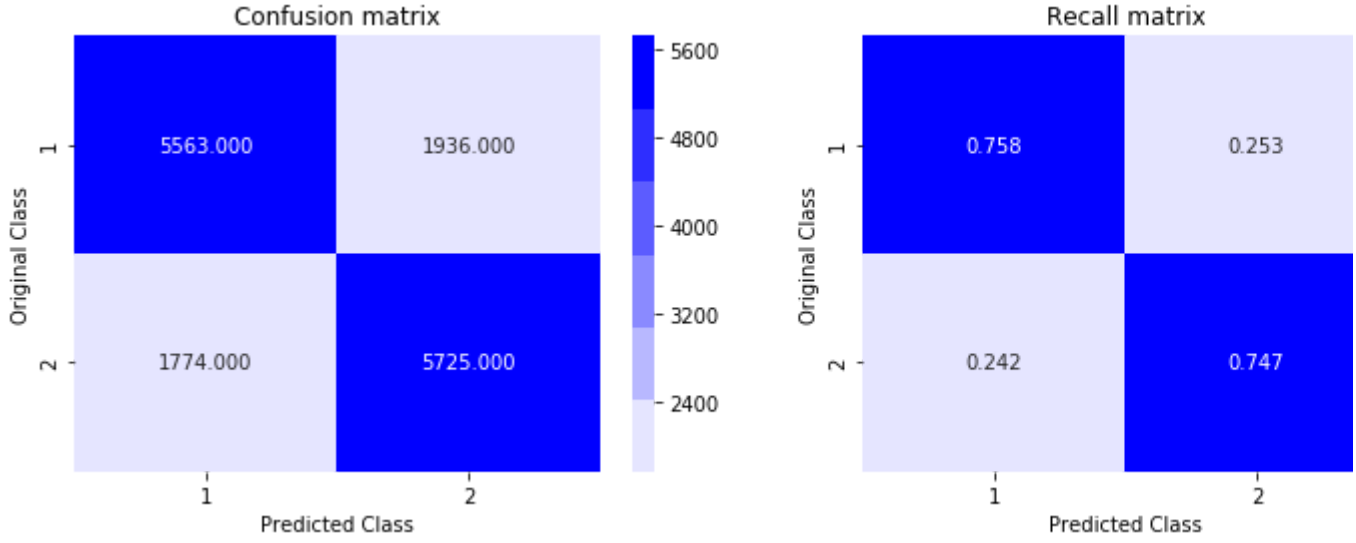
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X)  Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, class_weight = 'balanced
    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, labels=clf.cla

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, class_weight
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_loss(y_train, p
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
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: 0.6931471805599453
For values of alpha =  0.0001 The log loss is: 0.5091347858590833
For values of alpha =  0.001 The log loss is: 0.5199967097951628
For values of alpha =  0.01 The log loss is: 0.5338503762012379
For values of alpha =  0.1 The log loss is: 0.5112823402578778
For values of alpha =  1 The log loss is: 0.5151353602757957
For values of alpha =  10 The log loss is: 0.5990821400526668
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5076124311962193
For values of best alpha =  0.0001 The test log loss is: 0.5091347858590833
Total number of data points : 14998
```



Here the Recall(TPR) = 0.747 i.e., out of total positive points 74.7% are correctly predicted.

Recall = TP/(TP + FN)

Precision = 0.763 i.e., out of total points for which the model predicted as positive class 76.3% are true.

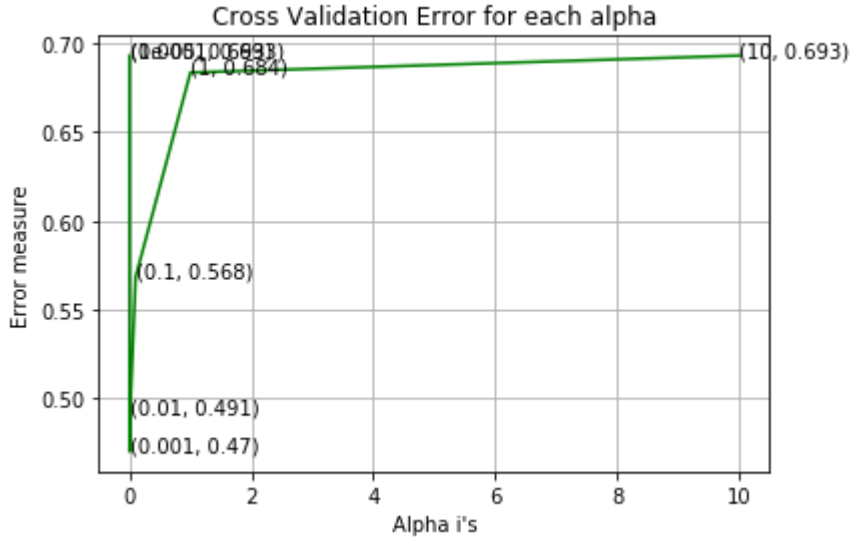
Precision = TP/(TP + FP)

TNR = 0.758 which seems the model is performing well for the negative points.

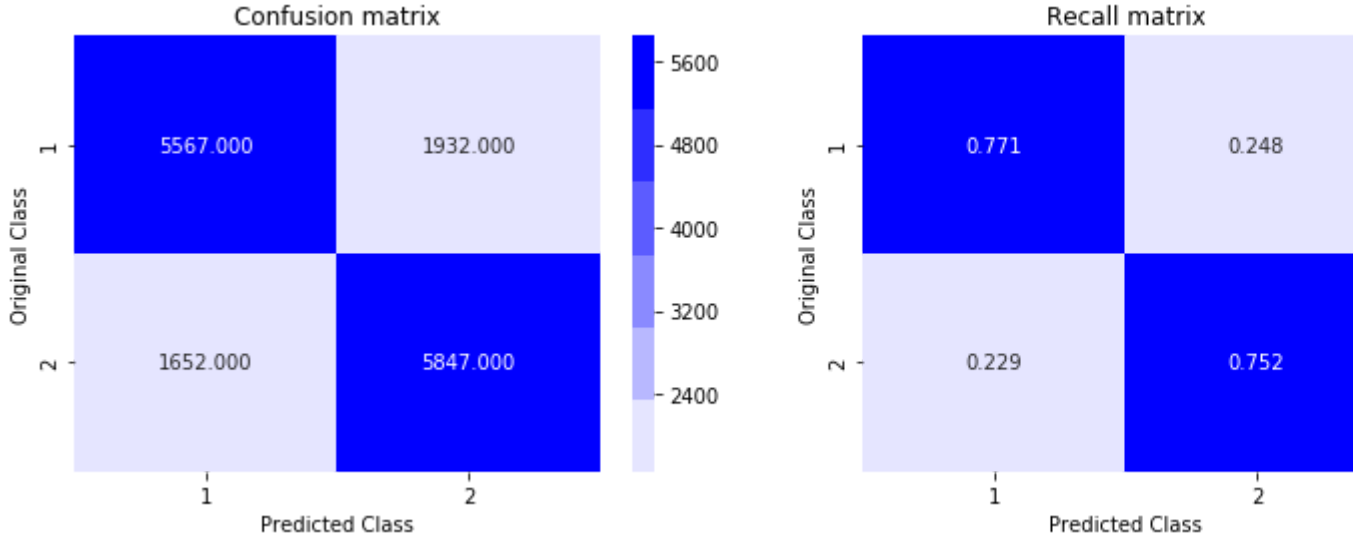
Linear SVM with hyperparameter tuning (TFIDF weighted W2V)

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)  Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42, class_weight = 'balanc
    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, labels=clf.cla

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=42, class_weig
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_loss(y_train, p
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
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: 0.6931471805599453
For values of alpha =  0.0001 The log loss is: 0.6931471805599453
For values of alpha =  0.001 The log loss is: 0.46991012224646067
For values of alpha =  0.01 The log loss is: 0.49142282949650967
For values of alpha =  0.1 The log loss is: 0.5682638211856019
For values of alpha =  1 The log loss is: 0.6835963352304967
For values of alpha =  10 The log loss is: 0.6931471805599453
```




```
For values of best alpha =  0.001 The train log loss is: 0.4698501197334016
For values of best alpha =  0.001 The test log loss is: 0.46991012224646067
Total number of data points : 14998
```



Here the Recall(TPR) = 0.752 i.e., out of total positive points 75.2% are correctly predicted.

Recall = TP/(TP + FN)

Precision = 0.780 i.e., out of total points for which the model predicted as positive class 78% are true.

Precision = TP/(TP + FP)

TNR = 0.771 which seems the model is performing well for the negative points.

4.6 XGBoost with parameter tuning (RandomSearch)

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV,StratifiedKFold

n_estimators = [100, 300, 500, 700, 900, 1100, 1300, 1500]
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
colsample_bytree = [0.1, 0.3, 0.5, 0.7 , 0.9, 1]
subsample = [0.1,0.3,0.5,0.7,0.9,1]

def hyperparameter_tunning(X,Y):
    param_grid = dict(learning_rate=learning_rate,
                      n_estimators=n_estimators,
                      colsample_bytree = colsample_bytree,
                      subsample = subsample)

    model = XGBClassifier(nthread=-1)
    kfold = StratifiedKFold(n_splits=5, shuffle=True)
    random_search = RandomizedSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfol
    random_result = random_search.fit(X,Y)

    # Summarize results
    print("Best: %f using %s" % (random_result.best_score_, random_result.best_params_))
    print()
    means = random_result.cv_results_['mean_test_score']
    stds = random_result.cv_results_['std_test_score']
    params = random_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))

    return random_result



start = dt.datetime.now()

# Tune hyperparameter values
random_result = hyperparameter_tunning(X_train,y_train)

print("\nTimeTaken: ",dt.datetime.now() - start)
```

```
Best: -0.345903 using {'subsample': 0.5, 'n_estimators': 1100, 'learning_rate': 0.1, 'co

-0.365879 (0.004230) with: {'subsample': 0.3, 'n_estimators': 1500, 'learning_rate': 0.1
-0.356139 (0.003697) with: {'subsample': 0.7, 'n_estimators': 700, 'learning_rate': 0.2,
-0.359284 (0.003443) with: {'subsample': 0.5, 'n_estimators': 300, 'learning_rate': 0.2,
-0.357010 (0.005302) with: {'subsample': 0.9, 'n_estimators': 900, 'learning_rate': 0.2,
-0.689719 (0.000032) with: {'subsample': 0.9, 'n_estimators': 100, 'learning_rate': 0.00
-0.689803 (0.000031) with: {'subsample': 0.5, 'n_estimators': 100, 'learning_rate': 0.00
-0.345903 (0.003506) with: {'subsample': 0.5, 'n_estimators': 1100, 'learning_rate': 0.1
-0.689690 (0.000033) with: {'subsample': 0.7, 'n_estimators': 100, 'learning_rate': 0.00
-0.686953 (0.000062) with: {'subsample': 0.5, 'n_estimators': 300, 'learning_rate': 0.00
-0.531286 (0.012739) with: {'subsample': 0.1, 'n_estimators': 900, 'learning_rate': 0.2,

TimeTaken:  0:33:20.760847
```

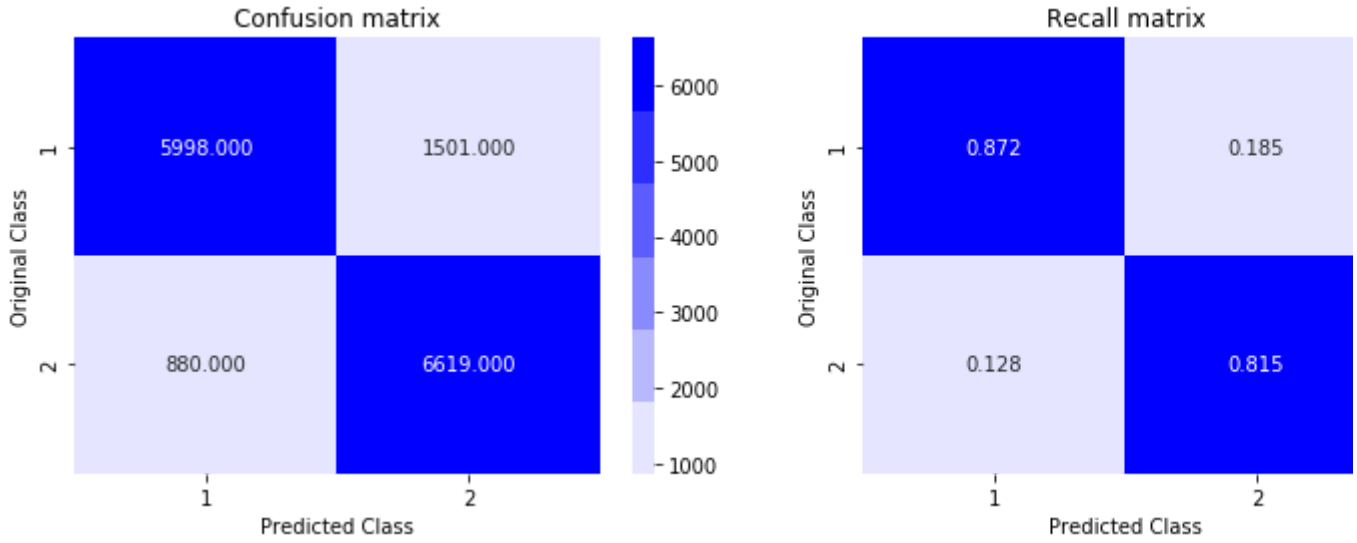
```
import xgboost as xgb

params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 3
params['colsample_bytree'] = 0.7
params['n_estimators'] = 500
```

```
params['subsample'] = 1
params['learning_rate'] = 0.3
params['nthread'] = -1
params['silent'] = 1

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,verbose_eval= False,early_stopping_rounds=20)

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-15))
print("\nTime Taken: ",dt.datetime.now() - start)
```

```
The test log loss is: 0.3415650370617024

Time Taken:  0:35:05.913193
```

```
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)
```

```
Total number of data points : 14998
```



The Hyperparameter tuning is done for the XGBOOST classifier where colsample_bytree, n_estimators, learning_ra hyperparameters.

Here the Recall(TPR) = 0.815 i.e., out of total positive points 81.5% are correctly predicted.

Recall = TP/(TP + FN)

Precision = 0.883 i.e., out of total points for which the model predicted as positive class 88.3% are true.

Precision = TP/(TP + FP)

Logistic Regression with hyperparameter tuning (TFIDF)

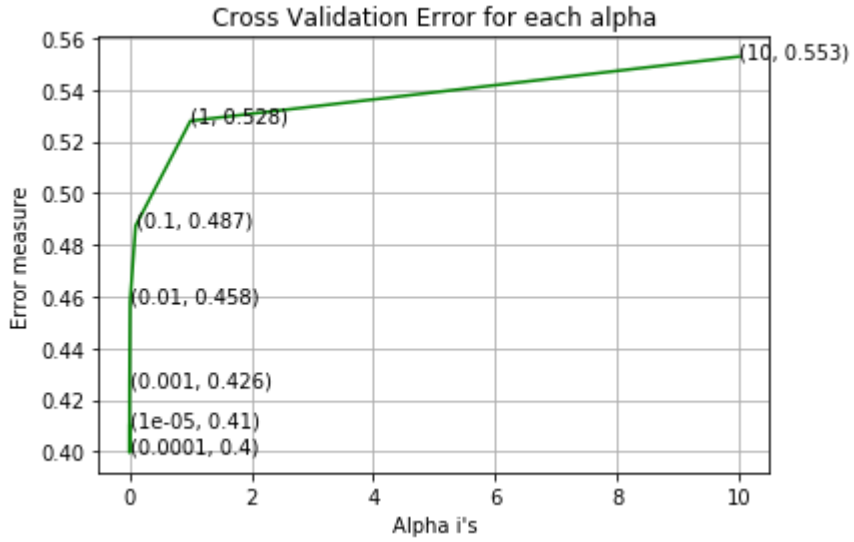
```
from scipy.sparse import hstack
```

```
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((X_train,q1_tfidf_train,q2_tfidf_train))
X_test = hstack((X_test,q1_tfidf_test,q2_tfidf_test))
X_train = X_train.tocsr()
X_test = X_test.tocsr()
print(X_train.shape , y_train.shape)
print(X_test.shape , y_test.shape)
```

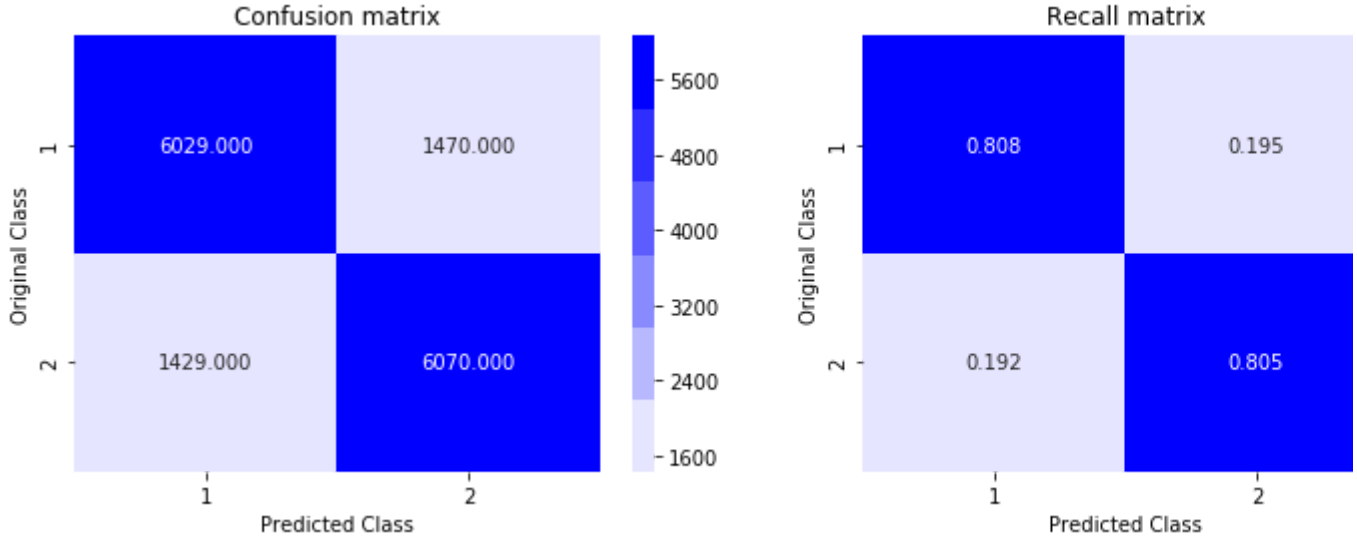
```
(34994, 6258) (34994,)
(14998, 6258) (14998,)
```

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)  Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, class_weight = 'balanced
    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, labels=clf.cla

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, class_weight
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_loss(y_train, p
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
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: 0.4100676187806122
For values of alpha =  0.0001 The log loss is: 0.399551468766388
For values of alpha =  0.001 The log loss is: 0.42563266600133476
For values of alpha =  0.01 The log loss is: 0.45792038066088564
For values of alpha =  0.1 The log loss is: 0.48733261803603195
For values of alpha =  1 The log loss is: 0.527933760890449
For values of alpha =  10 The log loss is: 0.5528078170011824
```




```
For values of best alpha =  0.0001 The train log loss is: 0.3653458108955217
For values of best alpha =  0.0001 The test log loss is: 0.399551468766388
Total number of data points : 14998
```



Here the Recall(TPR) = 0.805 i.e., out of total positive points 80.5% are correctly predicted.

Recall = TP/(TP + FN)

Precision = 0.809 i.e., out of total points for which the model predicted as positive class 80.9% are true.

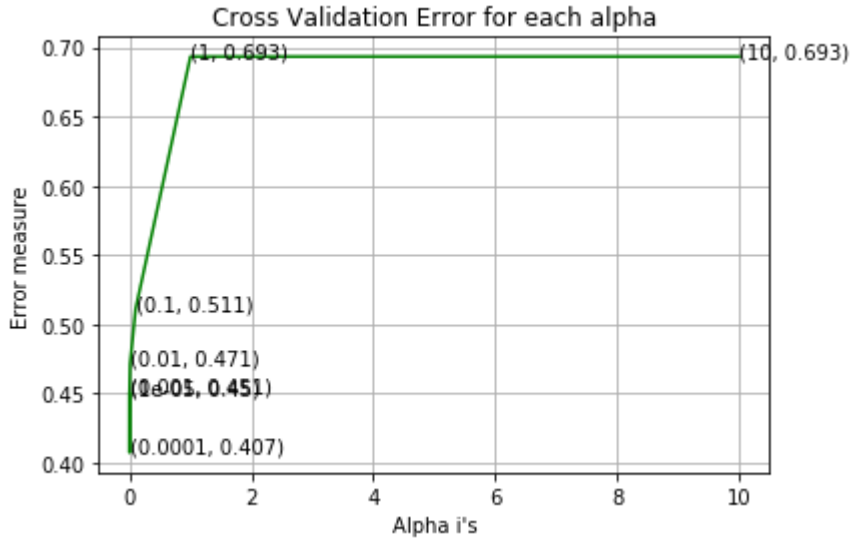
Precision = TP/(TP + FP)

TNR = 0.808 which seems the model is performing well for the negative points.

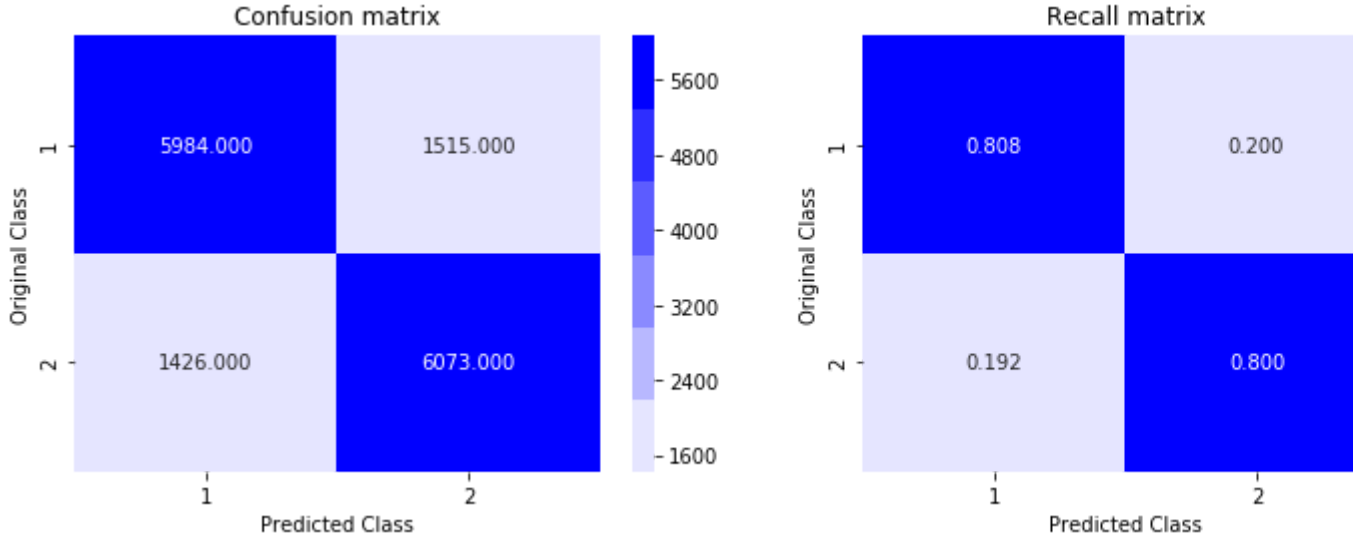
Linear SVM with hyperparameter tuning (TFIDF)

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)  Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42, class_weight = 'balanc
    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, labels=clf.cla

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=42, class_weig
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_loss(y_train, p
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
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: 0.45014217369546017
For values of alpha =  0.0001 The log loss is: 0.4074772303507898
For values of alpha =  0.001 The log loss is: 0.45088697522211557
For values of alpha =  0.01 The log loss is: 0.4714200638007782
For values of alpha =  0.1 The log loss is: 0.5110453797952003
For values of alpha =  1 The log loss is: 0.6931471805599453
For values of alpha =  10 The log loss is: 0.6931471805599453
```




```
For values of best alpha =  0.0001 The train log loss is: 0.3865680963204368
For values of best alpha =  0.0001 The test log loss is: 0.4074772303507898
Total number of data points : 14998
```



Here the Recall(TPR) = 0.800 i.e., out of total positive points 80.0% are correctly predicted.

Recall = TP/(TP + FN)

Precision = 0.810 i.e., out of total points for which the model predicted as positive class 81.0% are true.

Precision = TP/(TP + FP)

TNR = 0.808 which seems the model is performing well for the negative points