

3. Exploratory Data Analysis

```
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
import distance
from bs4 import BeautifulSoup
```



```
import time
import warnings
import sqlite3
#from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
from datetime import datetime as dt
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
import spacy
from tqdm import tqdm
from datetime import datetime as dt

```

3.1 Reading data and basic stats

```

df = pd.read_csv("train.csv",engine='python',encoding='utf-8',error_bad_lines=False)

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

```

```

❏ Skipping line 241459: unexpected end of data
   Number of data points: 241457

```

```
df.head()
```

```

❏

```

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

```
df.info()
```

```

❏ <class 'pandas.core.frame.DataFrame'>
   RangeIndex: 241457 entries, 0 to 241456
   Data columns (total 6 columns):
    #   Column          Non-Null Count  Dtype
---  -
0    id              241457 non-null  int64
1    qid1             241457 non-null  int64
2    qid2             241457 non-null  int64
3    question1        241457 non-null  object
4    question2        241455 non-null  object
5    is_duplicate     241457 non-null  int64
dtypes: int64(4), object(2)
memory usage: 11.1+ MB

```

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

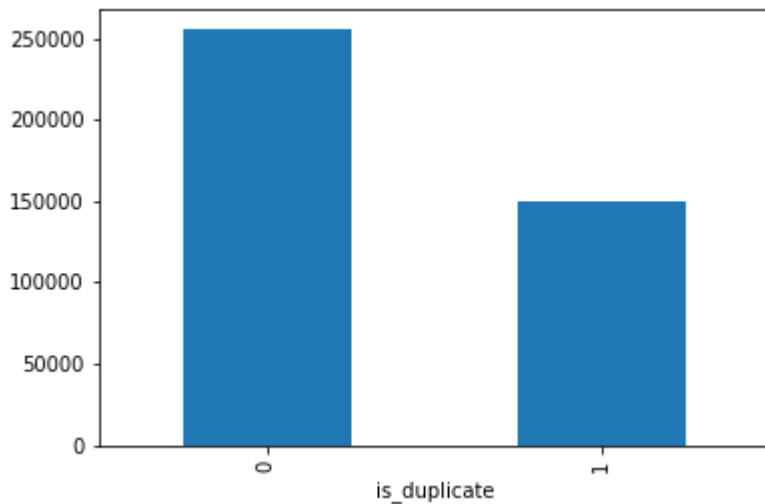
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

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

 <matplotlib.axes._subplots.AxesSubplot at 0x207cd59b128>



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

```
[> ~> Total number of question pairs for training:
      241457
```

```
df.duplicated().sum()
```

```
[> 0
```

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

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

```
[> ~> Question pairs are not Similar (is_duplicate = 0):
      62.79%
```

```
~> Question pairs are Similar (is_duplicate = 1):
      37.21%
```

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_onetime, (qs_morethan_onetime/len(qids)*100)))
```

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

```
q_vals=qids.value_counts()
```

```
q_vals=q_vals.values
```

```
q_vals=q_vals.values
```

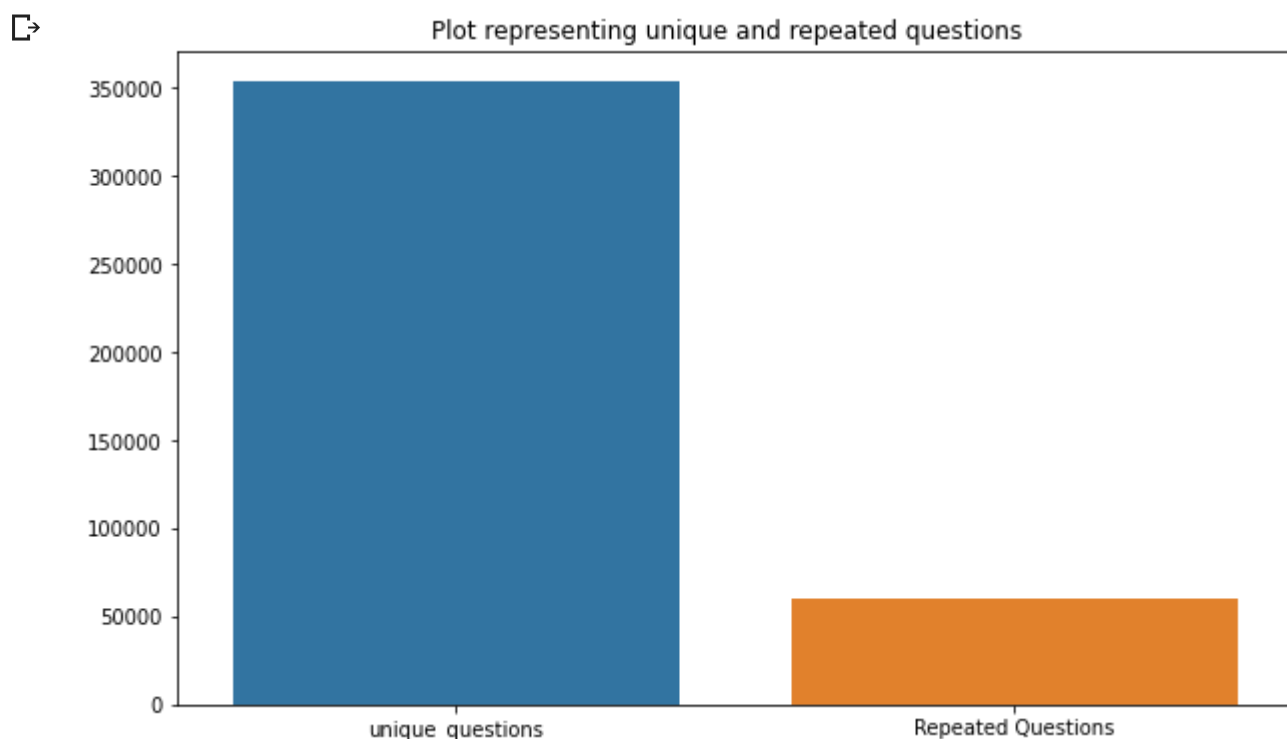
```
➞ Total number of Unique Questions are: 353463
```

Number of unique questions that appear more than one time: 60348 (17.073357041613974%)

Max number of times a single question is repeated: 89

```
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')
```

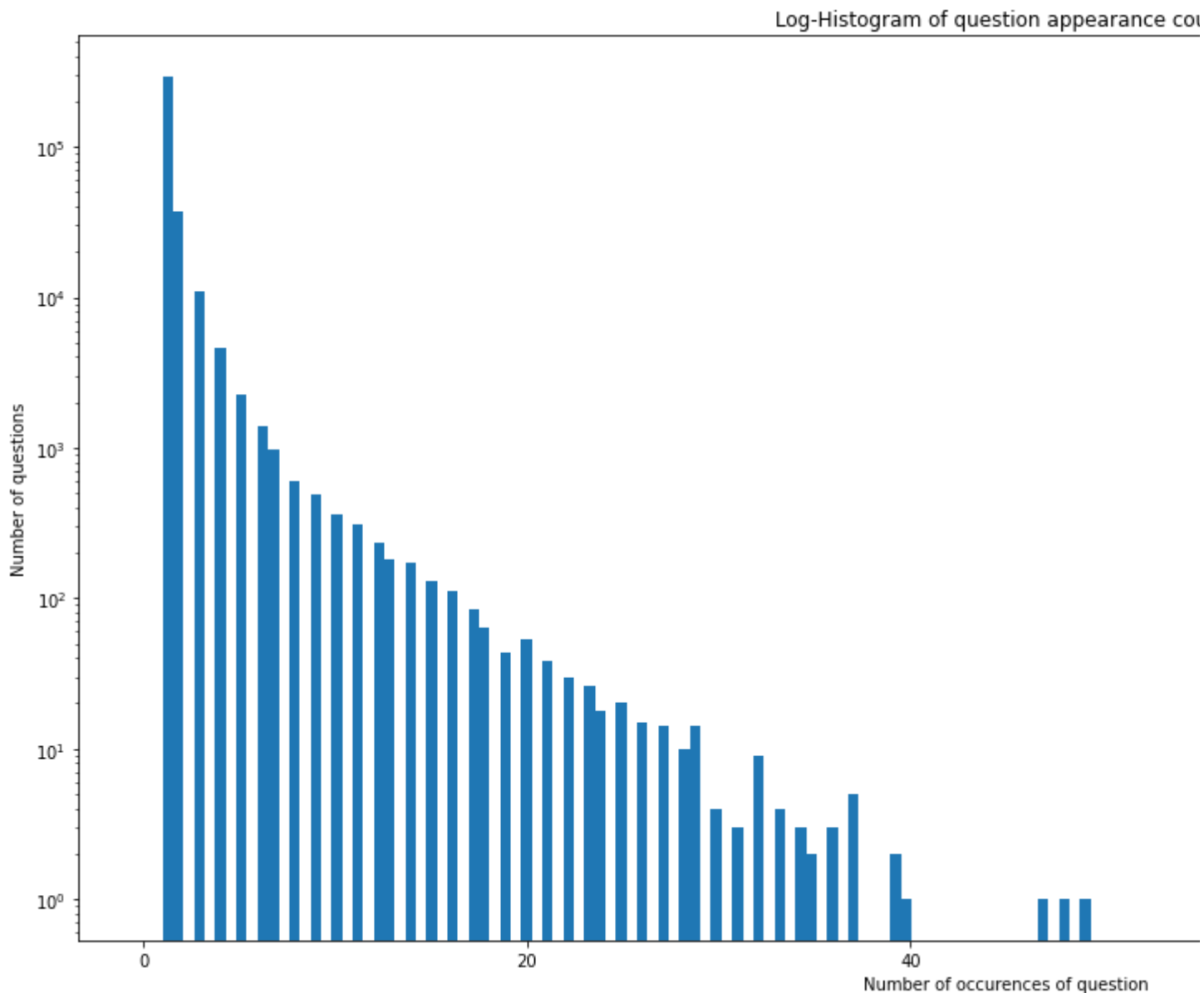
```
plt.title('Log-Histogram of question appearance counts')

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

plt.ylabel('Number of questions')

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

☞ Maximum number of times a single question is repeated: 89
```



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)
```

☞

	id	qid1	...	question2	is_duplicate
105780	105780	174363	...	NaN	0
201841	201841	303951	...	NaN	0

[2 rows x 6 columns]

- 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('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()
```



	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2
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	2
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	2	4
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	6
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when 23^{24} is divided by 100...	0	1	8
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	2	10

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] : 41
Number of Questions with minimum length [question2] : 19
```

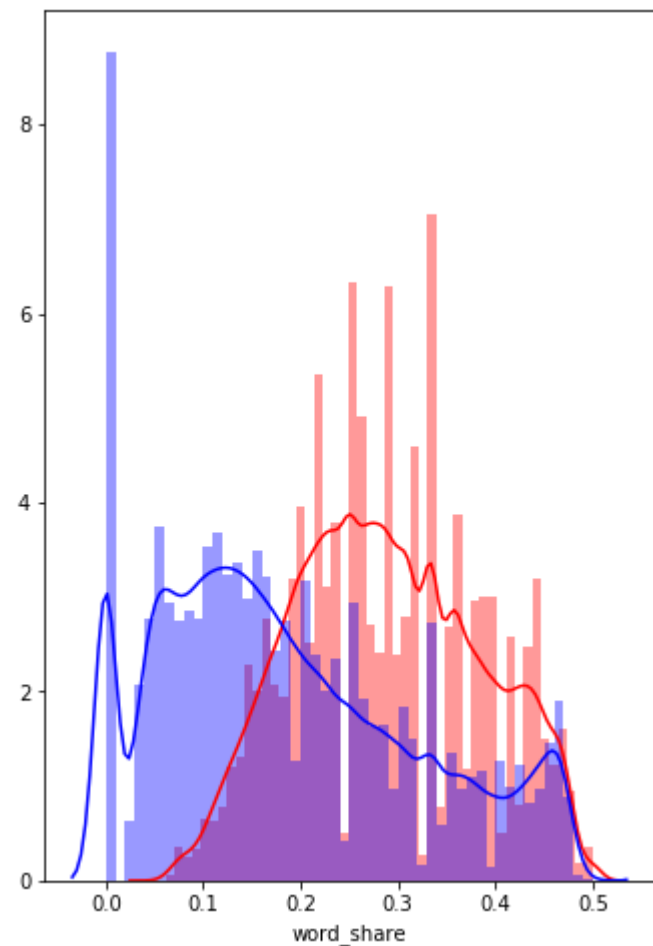
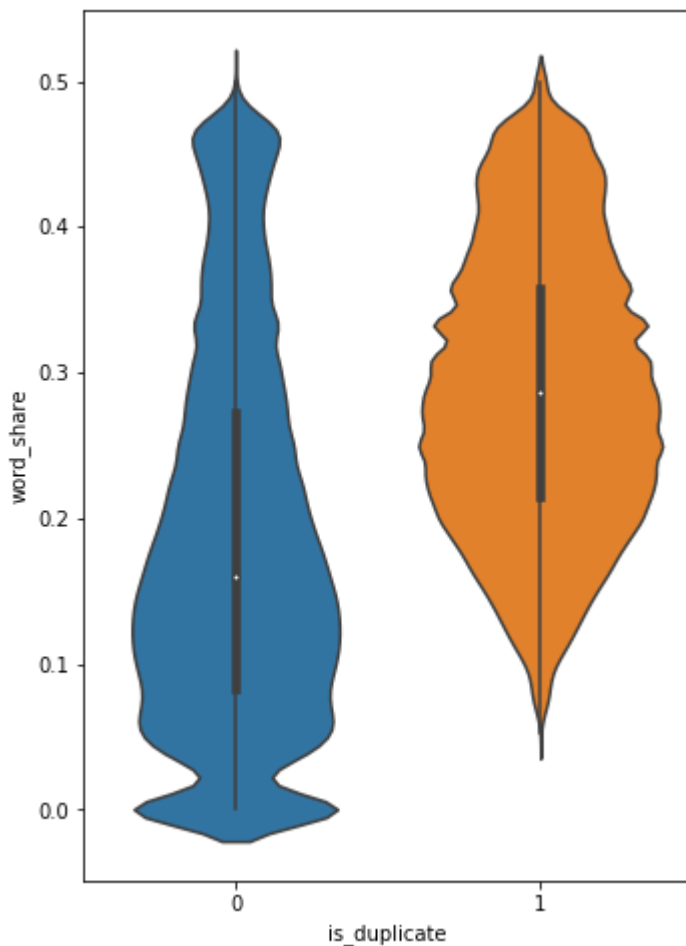
3.3.1.1 Feature: word_share

```
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()
```

```
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

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

```
plt.subplot(1,2,1)
```

```
sns.violinplot(x = 'is_duplicate', y = 'word_Common', data = df[0:])
```

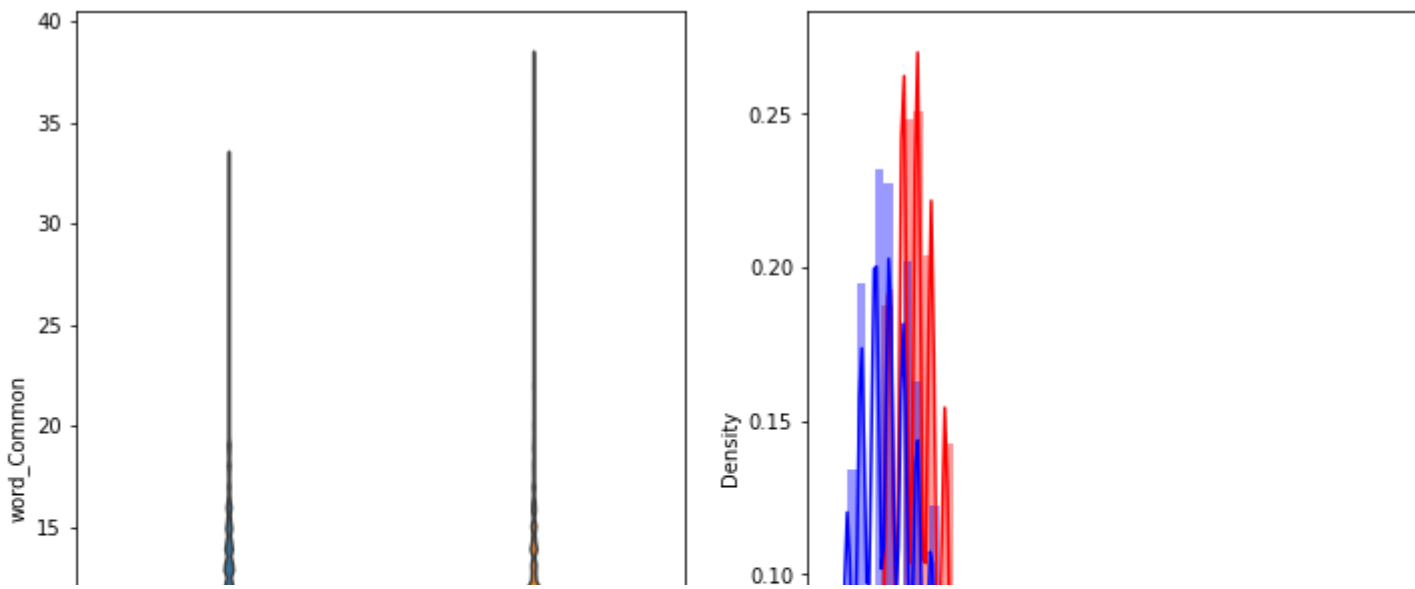
```
plt.subplot(1,2,2)
```

```
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



3.4 Preprocessing of Text



```
# To get the results in 4 decimal 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", "can not")\
        .replace("n't", " not").replace("what's", "what is").replace("it's", "it is")\
        .replace("'ve", " have").replace("i'm", "i am").replace("'re", " are")\
        .replace("he's", "he is").replace("she's", "she is").replace("'s", " own")\
        .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar")\
        .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)

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

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

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

    return x
```

```
return x

[> [nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
```

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

```
!pip install fuzzywuzzy
```

```
[> Collecting fuzzywuzzy
      Downloading https://files.pythonhosted.org/packages/43/ff/74f23998ad2f93b945c0309f825be92e04
      Installing collected packages: fuzzywuzzy
      Successfully installed fuzzywuzzy-0.18.0
```

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

import re
from nltk.corpus import stopwords
#import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
#import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

```
[>
```

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

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

    print("token features...")

    # Merging Features with dataset

```

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

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

#Computing Fuzzy Features and Merging with Dataset

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

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

```
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)
```



wc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff
999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2
799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5

▼ Analysis of extracted features

Plotting Word clouds

```

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

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

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

#Saving the np array into a text file
np.savetxt('train_p.txt', p, delimiter=' ', fmt='%s',encoding='utf-8')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s',encoding='utf-8')

```

```

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

```

```

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

```

```

textp_w = open(path.join(d, 'train_p.txt'), encoding="utf-8").read()
textn_w = open(path.join(d, 'train_n.txt'), encoding="utf-8").read()

```

```

stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

```

```

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

```

```

☞ Total number of words in duplicate pair questions : 16109886
   Total number of words in non duplicate pair questions : 33193067

```

```

wc = WordCloud(background_color="black", 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

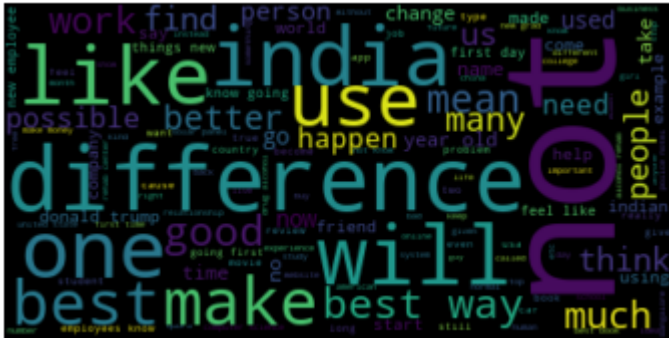


```

wc = WordCloud(background_color="black", 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()

```

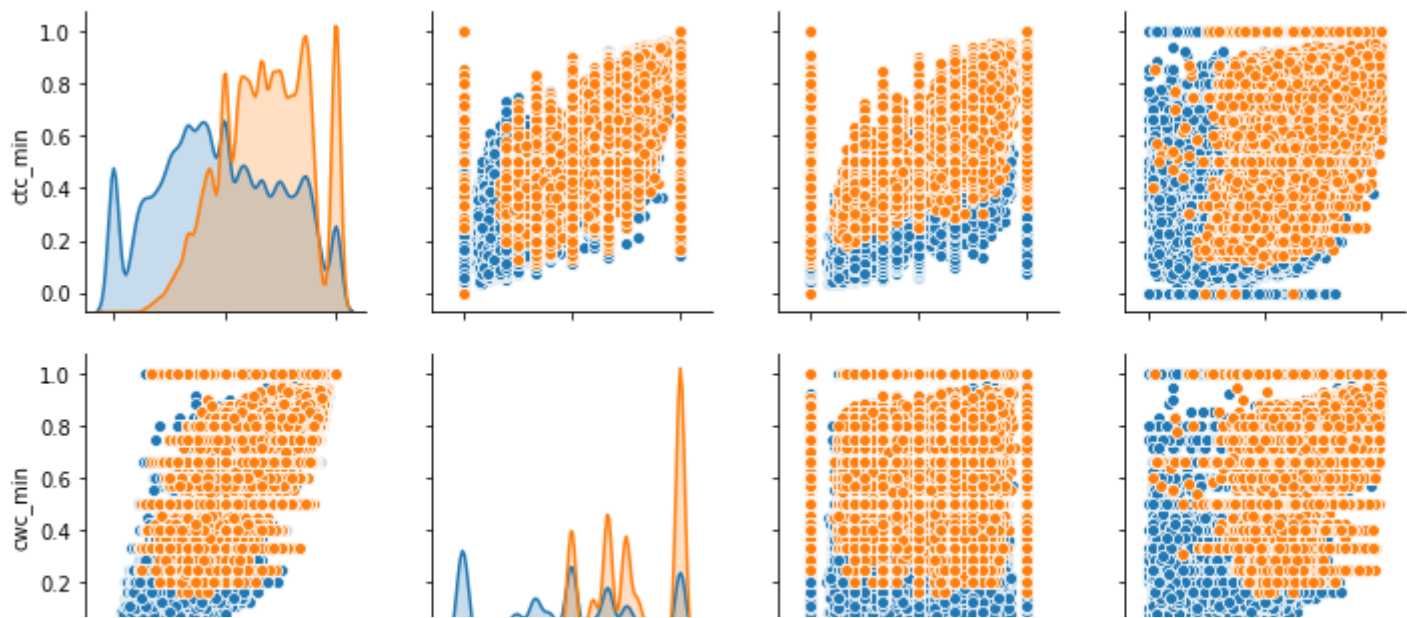
➡ Word Cloud for non-Duplicate Question pairs:



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

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





Distribution of the token_sort_ratio

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

```
plt.subplot(1,2,1)
```

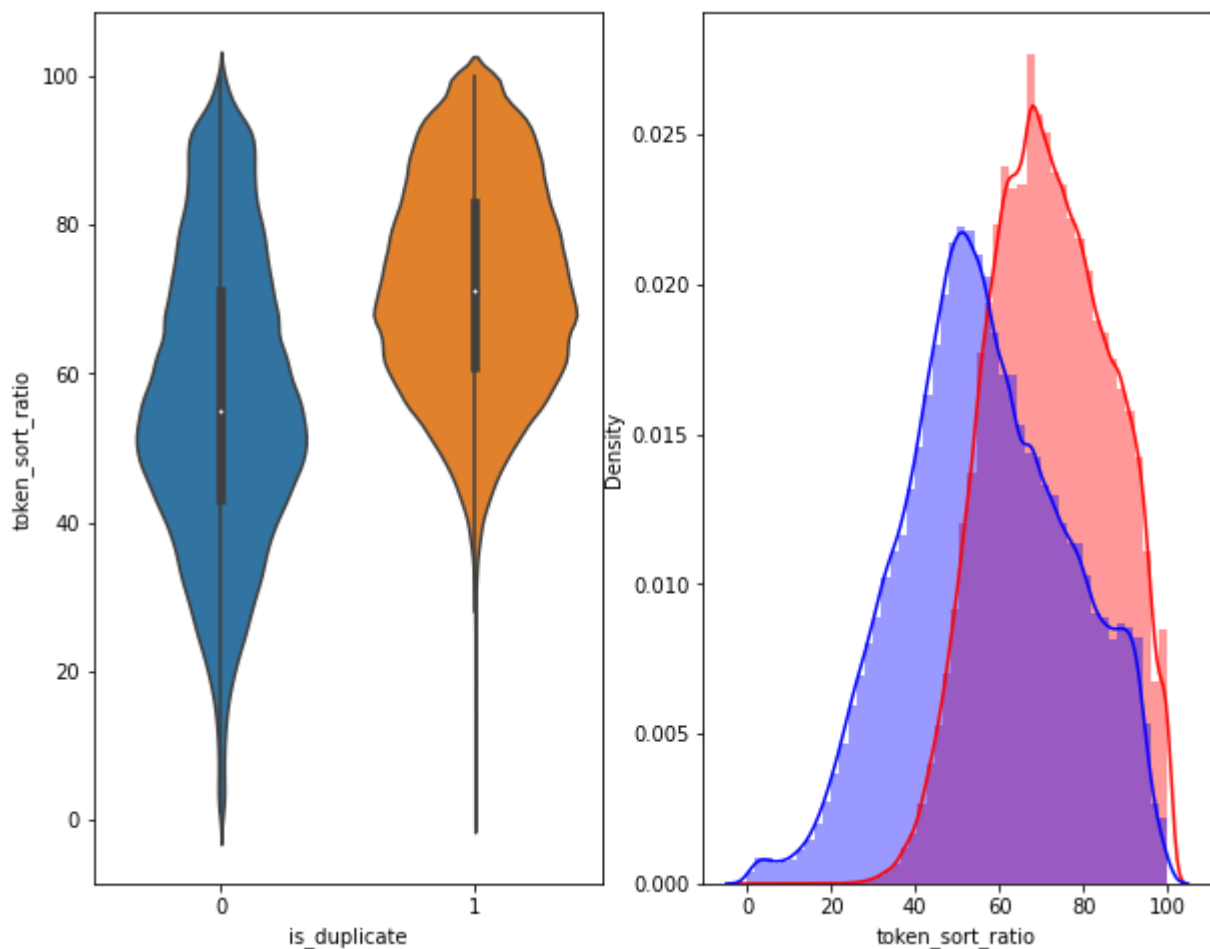
```
sns.violinplot(x = 'is_duplicate', y = 'token_sort_ratio', data = df[0:] , )
```

```
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()
```

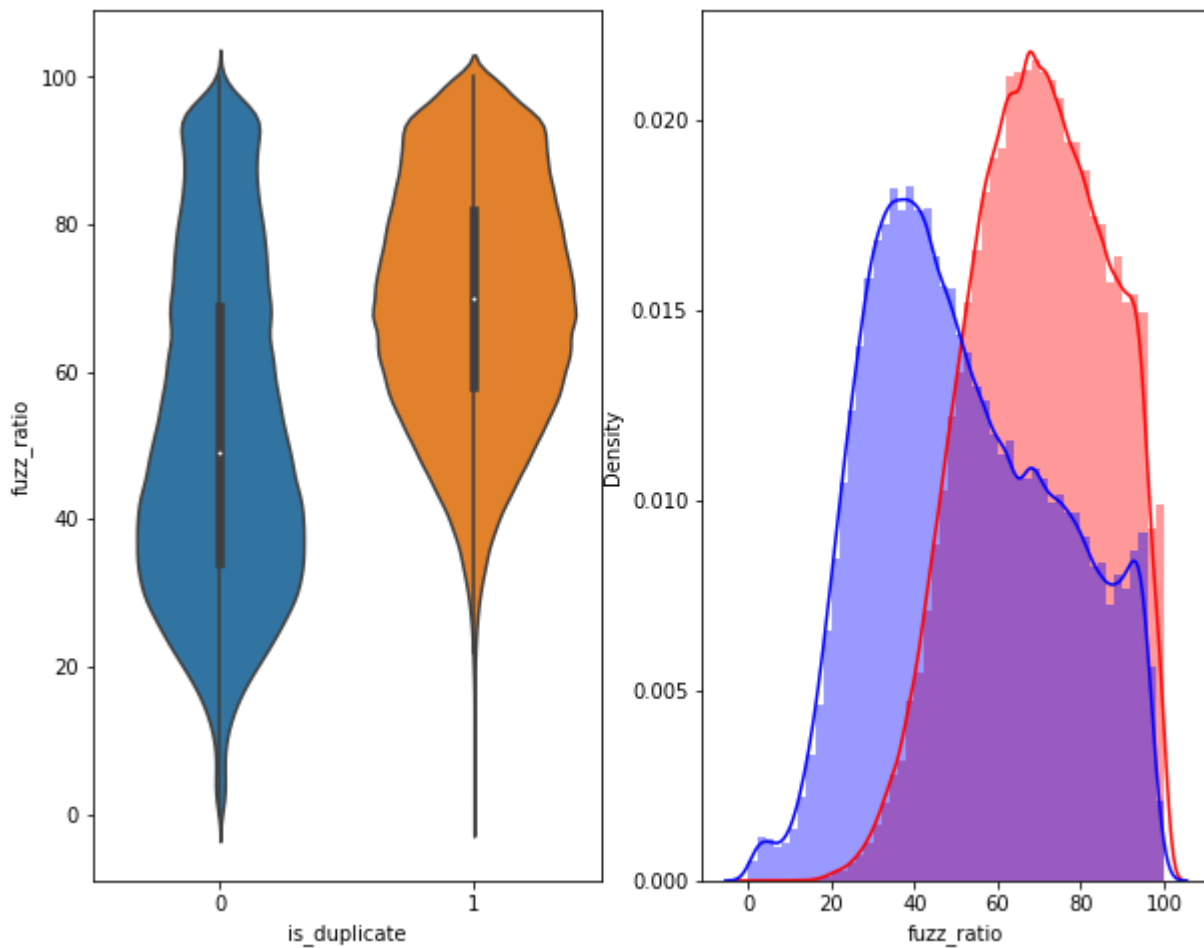


```
plt.figure(figsize=(10, 8))
```

```
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 Dimensionality reduction for 15 Features(Generated after cleaning the data) to 3 dimensions
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
dfp_subsampled = df[0:5000]
```

```
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max']])
```

```
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 0.032s...
[t-SNE] Computed neighbors for 5000 samples in 0.389s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.323s
[t-SNE] Iteration 50: error = 81.3346405, gradient norm = 0.0466835 (50 iterations in 2.500s)
[t-SNE] Iteration 100: error = 70.6411362, gradient norm = 0.0087385 (50 iterations in 1.740s)
[t-SNE] Iteration 150: error = 68.9421158, gradient norm = 0.0055224 (50 iterations in 1.692s)
[t-SNE] Iteration 200: error = 68.1217880, gradient norm = 0.0044136 (50 iterations in 1.712s)
[t-SNE] Iteration 250: error = 67.6154175, gradient norm = 0.0040027 (50 iterations in 1.801s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.615417
[t-SNE] Iteration 300: error = 1.7931896, gradient norm = 0.0011886 (50 iterations in 1.868s)
[t-SNE] Iteration 350: error = 1.3933632, gradient norm = 0.0004814 (50 iterations in 1.811s)
[t-SNE] Iteration 400: error = 1.2277179, gradient norm = 0.0002778 (50 iterations in 1.780s)
[t-SNE] Iteration 450: error = 1.1382203, gradient norm = 0.0001874 (50 iterations in 1.818s)
[t-SNE] Iteration 500: error = 1.0834213, gradient norm = 0.0001423 (50 iterations in 1.813s)
[t-SNE] Iteration 550: error = 1.0472572, gradient norm = 0.0001143 (50 iterations in 1.817s)
[t-SNE] Iteration 600: error = 1.0229475, gradient norm = 0.0000992 (50 iterations in 1.884s)
[t-SNE] Iteration 650: error = 1.0064161, gradient norm = 0.0000887 (50 iterations in 1.843s)
[t-SNE] Iteration 700: error = 0.9950126, gradient norm = 0.0000781 (50 iterations in 1.840s)
[t-SNE] Iteration 750: error = 0.9863916, gradient norm = 0.0000739 (50 iterations in 1.860s)
[t-SNE] Iteration 800: error = 0.9797955, gradient norm = 0.0000678 (50 iterations in 1.861s)
[t-SNE] Iteration 850: error = 0.9741892, gradient norm = 0.0000626 (50 iterations in 1.881s)
[t-SNE] Iteration 900: error = 0.9692684, gradient norm = 0.0000620 (50 iterations in 1.918s)
[t-SNE] Iteration 950: error = 0.9652601, gradient norm = 0.0000550 (50 iterations in 1.860s)

```

```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] , 'label':y})
```

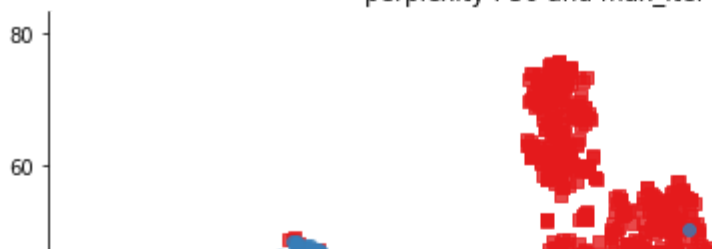
```

# draw the plot in appropriate place in the grid
sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()

```



perplexity : 30 and max_iter : 1000



```
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 0.024s...  
[t-SNE] Computed neighbors for 5000 samples in 0.390s...  
[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: 0.130446  
[t-SNE] Computed conditional probabilities in 0.312s  
[t-SNE] Iteration 50: error = 80.5661621, gradient norm = 0.0296227 (50 iterations in 10.168s)  
[t-SNE] Iteration 100: error = 69.4089432, gradient norm = 0.0033432 (50 iterations in 4.832s)  
[t-SNE] Iteration 150: error = 67.9962845, gradient norm = 0.0018752 (50 iterations in 4.428s)  
[t-SNE] Iteration 200: error = 67.4377289, gradient norm = 0.0011330 (50 iterations in 4.406s)  
[t-SNE] Iteration 250: error = 67.1244202, gradient norm = 0.0008592 (50 iterations in 4.444s)  
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.124420  
[t-SNE] Iteration 300: error = 1.5177890, gradient norm = 0.0007072 (50 iterations in 5.850s)  
[t-SNE] Iteration 350: error = 1.1818613, gradient norm = 0.0001967 (50 iterations in 7.515s)  
[t-SNE] Iteration 400: error = 1.0382802, gradient norm = 0.0000992 (50 iterations in 7.393s)  
[t-SNE] Iteration 450: error = 0.9668908, gradient norm = 0.0000785 (50 iterations in 7.300s)  
[t-SNE] Iteration 500: error = 0.9298934, gradient norm = 0.0000514 (50 iterations in 7.228s)  
[t-SNE] Iteration 550: error = 0.9096302, gradient norm = 0.0000429 (50 iterations in 7.108s)  
[t-SNE] Iteration 600: error = 0.8966513, gradient norm = 0.0000378 (50 iterations in 7.179s)  
[t-SNE] Iteration 650: error = 0.8874955, gradient norm = 0.0000321 (50 iterations in 7.152s)  
[t-SNE] Iteration 700: error = 0.8796885, gradient norm = 0.0000325 (50 iterations in 7.134s)  
[t-SNE] Iteration 750: error = 0.8725138, gradient norm = 0.0000287 (50 iterations in 7.081s)  
[t-SNE] Iteration 800: error = 0.8659297, gradient norm = 0.0000291 (50 iterations in 10.954s)  
[t-SNE] Iteration 850: error = 0.8608947, gradient norm = 0.0000276 (50 iterations in 7.138s)  
[t-SNE] Iteration 900: error = 0.8567888, gradient norm = 0.0000279 (50 iterations in 7.085s)  
[t-SNE] Iteration 950: error = 0.8539276, gradient norm = 0.0000273 (50 iterations in 7.033s)  
[t-SNE] Iteration 1000: error = 0.8515787, gradient norm = 0.0000235 (50 iterations in 7.078s)  
[t-SNE] KL divergence after 1000 iterations: 0.851579
```

```
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')
```

```

        color_bar = 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')

```



▼ Featurizing text data with tfidf weighted word-vectors

```
import pandas as pd
```

```

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

```

```

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

```

```
df = pd.read_csv("train.csv")
```

```

df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))

```

```
df.head()
```

```

↳

```

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

```

%%time
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_))

```

```

↳ CPU times: user 9.51 s, sys: 209 ms, total: 9.72 s
Wall time: 9.74 s

```

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

```
import en_core_web_sm
nlp=en_core_web_sm.load()
```

```
vecs=[]
```

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
```

```
import en_core_web_sm
nlp = en_core_web_sm.load()
#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%|██████████| 404290/404290 [1:05:59<00:00, 102.11it/s]
```

```
vecs2 = []
```

```
for qu2 in tqdm(list(df['question2'])):
```

```
    doc2 = nlp(qu2)
```

```
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
```

```
    for word2 in doc2:
```

```
        # word2vec
```

```
        vec2 = word2.vector
```

```
        # fetch df score
```

```
        try:
```

```
            idf = word2tfidf[str(word2)]
```

```
        except:
```

```
            #print word
```

```
            idf = 0
```

```
        # compute final vec
```

```
        mean_vec2 += vec2 * idf
```

```
    mean_vec2 = mean_vec2.mean(axis=0)
```

```
    vecs2.append(mean_vec2)
```

```
df['q2_feats_m'] = list(vecs2)
```

```
df[q2_feats_] = list(vecs2)
```

```
100%|██████████| 404290/404290 [1:06:12<00:00, 101.77it/s]
```

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
```

```
#nlp_features_train.csv (NLP Features)
```

```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
```

```
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
```

```
#nlp_features_train.csv (NLP Features)
```

```
if os.path.isfile('train.csv'):
```

```
    dfnlp = pd.read_csv("train.csv",nrows=50000,encoding='latin-1')
```

```
# dataframe of nlp features
```

```
dfnlp.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 to invest in sh...
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian governm...

```
# data before preprocessing
```

```
dfppro.head(2)
```

```
id  qid1  qid2  question1  question2  is_duplicate  freq_qid1  freq_qid2
```

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	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	2

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
```

```
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

```
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

```
df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)
```

```
df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
```

```
# Questions 1 tfidf weighted word2vec
```

```
df3_q1.head()
```

	0	1	2	3	4	5	6	7
# Questions 2 tfidf weighted word2vec								
df3_q2.head()								
	0	1	2	3	4	5	6	7
0	-14.616981	59.755488	-53.263745	19.514497	113.916473	101.657056	8.561499	66.232769
1	-3.565742	-16.844571	-130.911785	0.320254	79.350278	23.562028	79.124551	84.119839
2	156.833630	59.991896	-8.414311	29.251426	133.680218	112.457566	89.849781	21.613022
3	41.472439	56.717317	31.530616	-5.520164	33.454800	79.596179	15.508996	40.042066
4	-14.446975	-4.338255	-70.196208	-48.636382	18.356858	-50.807069	24.311196	60.043674

5 rows × 96 columns

```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v dataframe :", df3_q1.shape[1])
print("Number of features in question2 w2v dataframe :", df3_q2.shape[1])
print("Number of features in final dataframe :", df1.shape[1]+df2.shape[1]+df3_q1.shape[1]+df3_q2.shape[1])
```

```
Number of features in nlp dataframe : 2
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v dataframe : 96
Number of features in question2 w2v dataframe : 96
Number of features in final dataframe : 206
```

```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1['id']=df1['id']
    df3_q2['id']=df1['id']
    df1 = df1.merge(df2, on='id',how='left')
    # df2 = df3_q1.merge(df3_q2, on='id',how='left')
    result = df1.merge(df2, on='id',how='left')
    result.to_csv('final_features.csv')
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

```
if os.path.isfile('final_features.csv'):
    data = pd.read_csv('final_features.csv',nrows=50000,encoding='utf-8')
```

```
data.head(3)
```



	Unnamed: 0	id	is_duplicate	freq_qid1_x	freq_qid2_x	q1len_x	q2len_x	q1_n_words_x	q2_n_words_x
0	0	0	0	1	1	66	57	14	14
1	1	1	0	2	1	51	88	8	8
2	2	2	0	1	1	73	59	14	14

4.3 Random train test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,data['is_duplicate'],stratify=data['is_duplicate'],test_size=0.2)
```

```
X_train.shape
```

```
(37500, 25)
```

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

```
(37500, 25)
```

	Unnamed: 0	id	is_duplicate	freq_qid1_x	freq_qid2_x	q1len_x	q2len_x	q1_n_words_x	q2_n_words_x
23561	23561	23561	0	1	1	33	50	7	7
3536	3536	3536	0	1	1	46	58	10	10

```
# extraction features from train data frame
X_train = X_train.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace=False)
```

```
# extraction features from test data frame
X_test = X_test.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace=False)
```

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

```
Number of data points in train data : (37500, 22)
Number of data points in test data : (12500, 22)
```

```
y_train.shape
```

```
(37500,)
```

```
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 test data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[0])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
(37500,)
```



```

----- Distribution of output variable in train data -----
Class 0: 0.6270133333333333 Class 1: 0.3729866666666667
----- Distribution of output variable in test data -----

# 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 dimension
    # 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 dimension
    # 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("Precision 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("Recall matrix")

    plt.show()

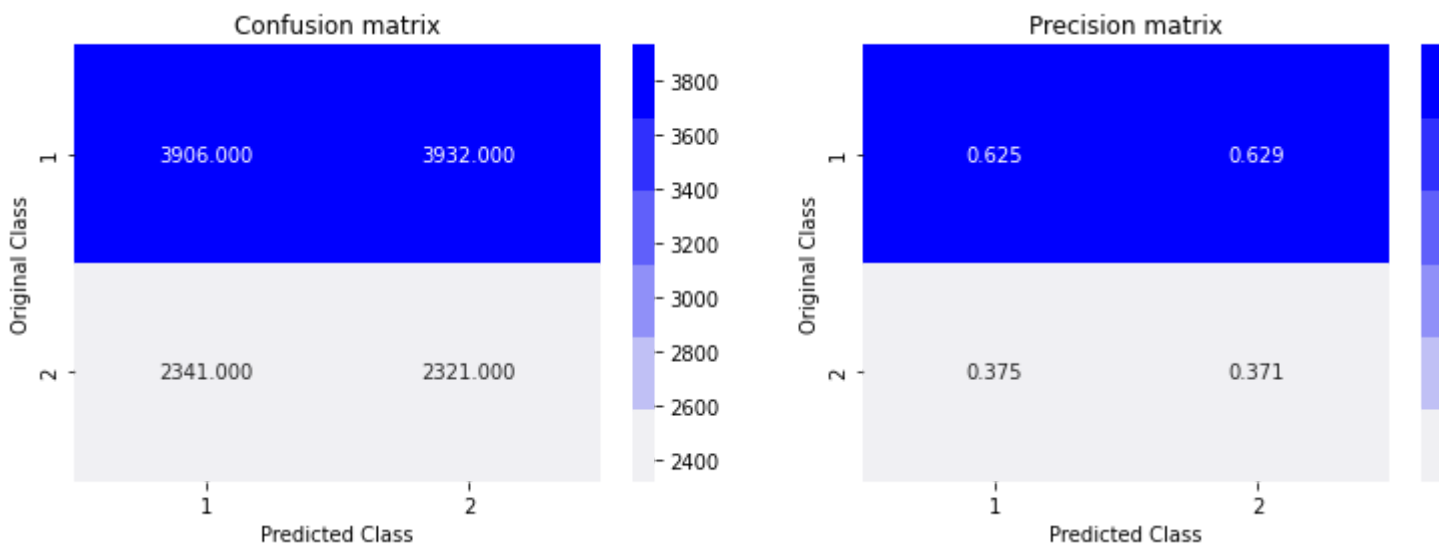
```

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

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs))))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-15))

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

Log loss on Test Data using Random Model 0.8788373475422782



4.4 Logistic Regression with hyperparameter tuning

```
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\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.01,
# 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.
```

```
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
```

```

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.classes_))

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()

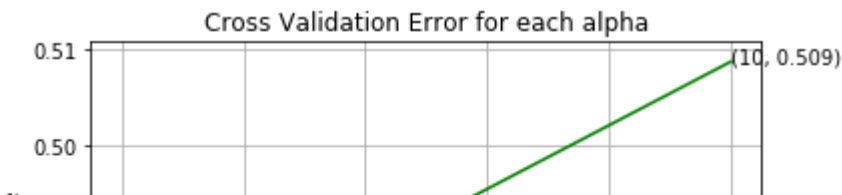
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

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, predict_y, labels=clf.classes_))
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, predict_y, labels=clf.classes_))
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: 0.46748950473787415
 For values of alpha = 0.0001 The log loss is: 0.467411411795257
 For values of alpha = 0.001 The log loss is: 0.4702043503620991
 For values of alpha = 0.01 The log loss is: 0.47095651464973903
 For values of alpha = 0.1 The log loss is: 0.47815931432174075
 For values of alpha = 1 The log loss is: 0.47883932764652315
 For values of alpha = 10 The log loss is: 0.5087726243467064



4.5 Linear SVM with hyperparameter tuning



```
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_iter=1000,
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.1,
```

```
# 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.
```

```
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, labels=clf.classes_, eps=1e-15))
```

```
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)
```

```
clf.fit(X_train, y_train)
```

```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

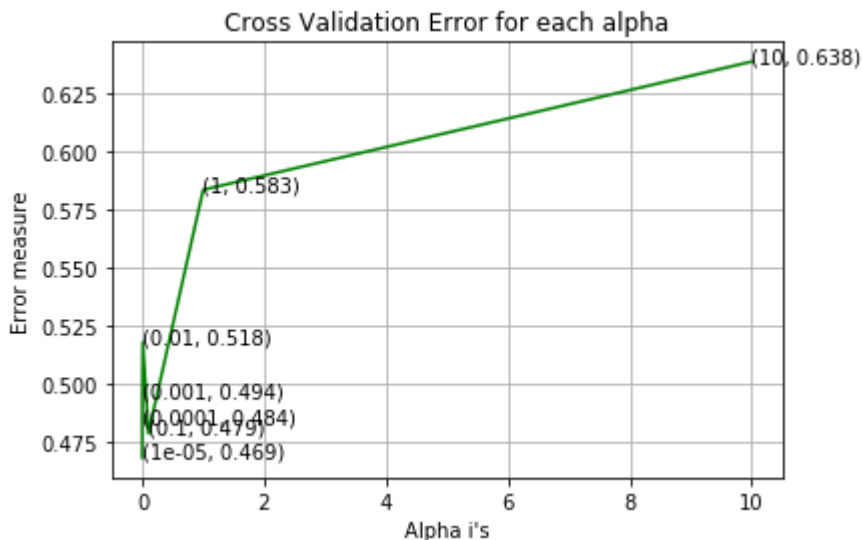
```
sig_clf.fit(X_train, y_train)
```

```

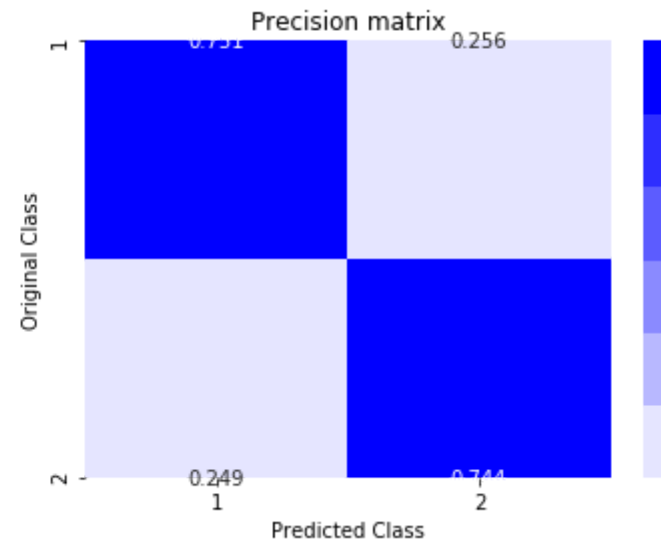
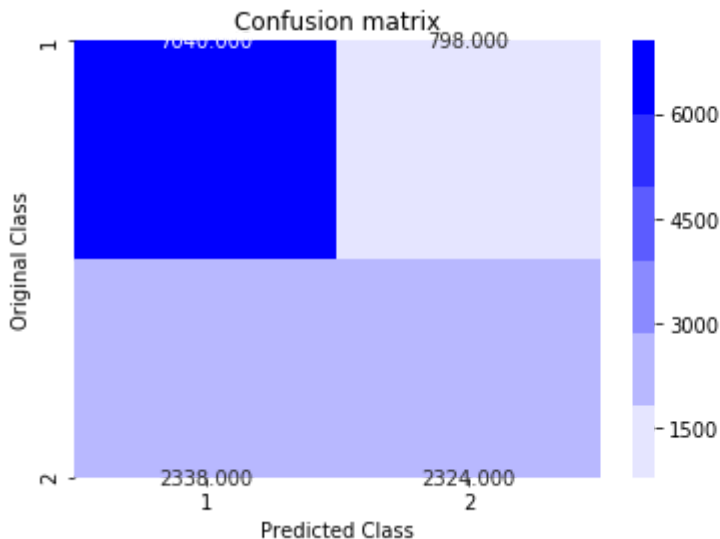
predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
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: 0.468568715140383
 For values of alpha = 0.0001 The log loss is: 0.4836764509430551
 For values of alpha = 0.001 The log loss is: 0.49419792841068927
 For values of alpha = 0.01 The log loss is: 0.5181667966968087
 For values of alpha = 0.1 The log loss is: 0.47892462615236553
 For values of alpha = 1 The log loss is: 0.583450288631332
 For values of alpha = 10 The log loss is: 0.6384048153029616



For values of best alpha = 1e-05 The train log loss is: 0.46792983199906313
 For values of best alpha = 1e-05 The test log loss is: 0.468568715140383
 Total number of data points : 12500



4.6 XGBoost

```

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

```

```

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

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

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, verbose_eval=10)

xgdmatrix = 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))

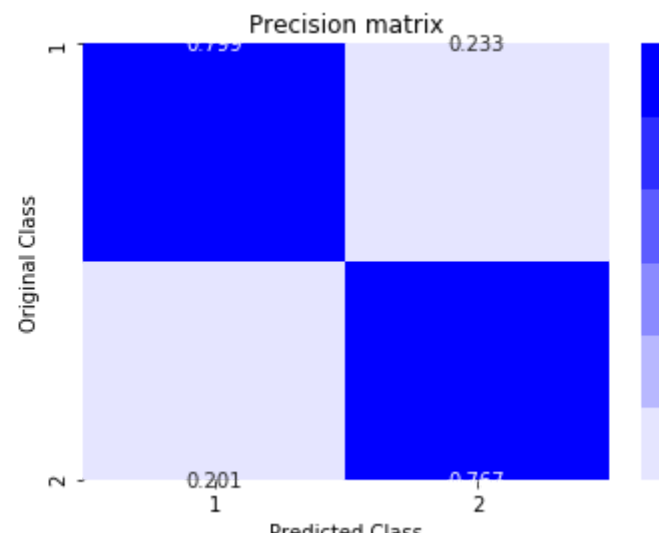
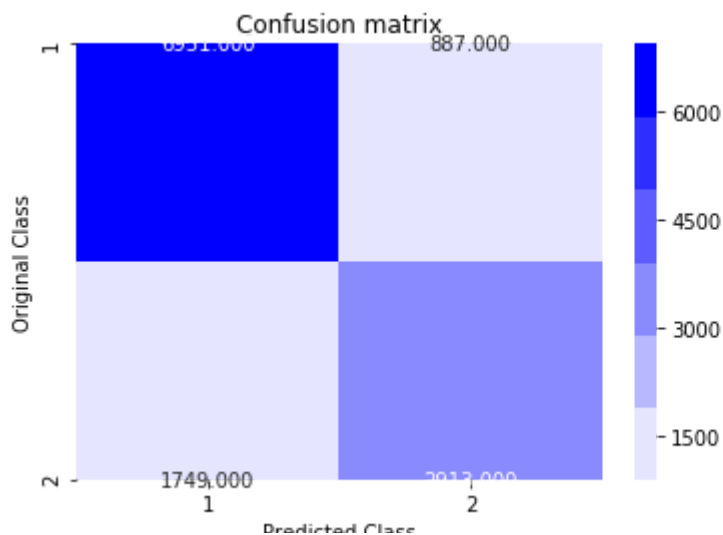
[0]      train-logloss:0.685301  valid-logloss:0.685242
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]      train-logloss:0.620617  valid-logloss:0.620098
[20]      train-logloss:0.573226  valid-logloss:0.572814
[30]      train-logloss:0.537786  valid-logloss:0.537428
[40]      train-logloss:0.511344  valid-logloss:0.51081
[50]      train-logloss:0.490437  valid-logloss:0.489951
[60]      train-logloss:0.475004  valid-logloss:0.47453
[70]      train-logloss:0.462591  valid-logloss:0.462188
[80]      train-logloss:0.452563  valid-logloss:0.452189
[90]      train-logloss:0.4443    valid-logloss:0.443862
[100]     train-logloss:0.437586  valid-logloss:0.437173
[110]     train-logloss:0.432111  valid-logloss:0.431734
[120]     train-logloss:0.427581  valid-logloss:0.427275
[130]     train-logloss:0.423936  valid-logloss:0.423784
[140]     train-logloss:0.420928  valid-logloss:0.420948
[150]     train-logloss:0.418488  valid-logloss:0.418605
[160]     train-logloss:0.416269  valid-logloss:0.416503
[170]     train-logloss:0.414331  valid-logloss:0.414624
[180]     train-logloss:0.412715  valid-logloss:0.413083
[190]     train-logloss:0.411387  valid-logloss:0.411845
[200]     train-logloss:0.409991  valid-logloss:0.410484
[210]     train-logloss:0.408918  valid-logloss:0.409456
[220]     train-logloss:0.407908  valid-logloss:0.408576
[230]     train-logloss:0.406951  valid-logloss:0.407746
[240]     train-logloss:0.406023  valid-logloss:0.406964
[250]     train-logloss:0.405104  valid-logloss:0.40621
[260]     train-logloss:0.404181  valid-logloss:0.405476
[270]     train-logloss:0.403293  valid-logloss:0.404614
[280]     train-logloss:0.402551  valid-logloss:0.40399
[290]     train-logloss:0.401995  valid-logloss:0.403561
[300]     train-logloss:0.401437  valid-logloss:0.403178
[310]     train-logloss:0.4008    valid-logloss:0.402659
[320]     train-logloss:0.400363  valid-logloss:0.402358
[330]     train-logloss:0.399898  valid-logloss:0.402057
[340]     train-logloss:0.399432  valid-logloss:0.401745
[350]     train-logloss:0.39891    valid-logloss:0.401362
[360]     train-logloss:0.398474  valid-logloss:0.40103
[370]     train-logloss:0.398037  valid-logloss:0.400724
[380]     train-logloss:0.397639  valid-logloss:0.400436
[390]     train-logloss:0.397208  valid-logloss:0.400196
[399]     train-logloss:0.396794  valid-logloss:0.399874
The test log loss is: 0.399874138790532

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 : 12500



1. Let us 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.

5.1 Reading data from file

```
if os.path.isfile('nlp_features_train.csv'):
    df1 = pd.read_csv("nlp_features_train.csv",nrows=50000,encoding='latin-1')

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

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

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
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
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

```
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
dfnlp = df1.merge(df2, on='id',how='left')
```

```
dfnlp.head(2)
```



_ratio	token_sort_ratio	fuzz_ratio	fuzz_partial_ratio	longest_substr_ratio	freq_qid1	freq_qid2
--------	------------------	------------	--------------------	----------------------	-----------	-----------

100	93	93	100	0.982759	1	1
-----	----	----	-----	----------	---	---

86	63	66	75	0.596154	2	2
----	----	----	----	----------	---	---

```
nan_rows = dfnlp[dfnlp.isnull().any(1)]
print (nan_rows)
```

```

[5] id      qid1    qid2    ... word_share freq_q1+q2 freq_q1-q2
3306  3306    6553    6554    ...      0.0         2         0
13016 13016   25026   25027   ...      0.0         3         1
20072 20072   37898   37899   ...      0.0         2         0
20794 20794   39204   39205   ...      0.0         2         0
47056 47056   84067   84068   ...      0.0         3         1
```

```
[5 rows x 32 columns]
```

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

```

[5] Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate, cwc_min, cwc_max, csc_min, csc_max]
Index: []
```

5.2 Splitting data into Train and cross validation(or test): Stratified Sampling

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dfnlp,dfnlp['is_duplicate'],stratify=dfnlp['is_duplicate'])
```

```
X_train= X_train.drop(('is_duplicate'),axis=1)
X_train.shape
```

```
[5] (37500, 31)
```

```
y_train.shape
```

```
[5] (37500,)
```

```
y_test.shape
```

```
[5] (12500,)
```



```
X_test= X_test.drop(('is_duplicate'),axis=1)
X_test.shape

(12500, 31)
```

```
X_train.head()
```

	id	qid1	qid2	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ctc_n
23561	23561	44124	44125	how do i learn geography for nda	how do i learn to accept myself and my appeara...	0.333322	0.333322	0.749981	0.428565	0.5714
3536	3536	7006	7007	what happens when 0 gb disk space is reached	is there a pokemon fan game or romhack set dur...	0.000000	0.000000	0.333322	0.166664	0.1117
33192	33192	61018	19621	why do people ask so many googleable questions...	why do some people ask questions on quora that...	0.666656	0.399996	0.749981	0.374995	0.6999
35725	35725	65244	65245	what is china doing to help nepal	how can we help nepal	0.999950	0.666644	0.000000	0.000000	0.3999
6320	6320	12389	12390	what are the best education portals in india	which are the best sites for free education in...	0.749981	0.599988	0.749981	0.599988	0.7499

5.3 TFIDF vectorizer on Questions Text Data

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10)

# merge texts
questions = list(X_train['question1']) + list(X_train['question2'])

vectorizer.fit(questions)


```

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=None,
```

▼ Train Data

```
tfidf_train_ques1= vectorizer.transform(X_train['question1'])
print("Shape of matrix after one hot encodig ",tfidf_train_ques1.shape)

print("the number of unique words ", tfidf_train_ques1.get_shape()[1])
```

```
➤ Shape of matrix after one hot encodig (37500, 13369)
the number of unique words 13369
```

```
tfidf_train_ques2= vectorizer.transform(X_train['question2'])
print("Shape of matrix after one hot encodig ",tfidf_train_ques2.shape)
print("the number of unique words ", tfidf_train_ques2.get_shape()[1])
```

```
➤ Shape of matrix after one hot encodig (37500, 13369)
the number of unique words 13369
```

```
# extraction features from train data frame
X_train_feature_df = X_train.drop(['id','qid1','qid2','question1','question2'], axis=1, inplace=False)
```

```
X_train_feature_df.head(2)
```

```
➤
```

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	a
23561	0.333322	0.333322	0.749981	0.428565	0.57142	0.399996	0.0	1.0	
3536	0.000000	0.000000	0.333322	0.166664	0.11111	0.083333	0.0	0.0	

```
import scipy
# X_train.head()
print("train Shape Before -> ",X_train_feature_df.shape," Type",type(X_train_feature_df))
```

```
#we need to convert the feature data into sparse matrix so that we can combine our features and tfidf
train_feat_sparse = scipy.sparse.csr_matrix(X_train_feature_df)
```

```
print("train Shape After-> ",train_feat_sparse.shape," Type",type(train_feat_sparse))
```

```
➤ train Shape Before -> (37500, 26) Type <class 'pandas.core.frame.DataFrame'>
train Shape After-> (37500, 26) Type <class 'scipy.sparse.csr.csr_matrix'>
```

▼ TEST Data

```
tfidf_test_ques1= vectorizer.transform(X_test['question1'])
```

```
print("Shape of matrix after one hot encoding ",tfidf_test_ques1.shape)
print("the number of unique words ", tfidf_test_ques1.get_shape()[1])
```

```
tfidf_test_ques2= vectorizer.transform(X_test['question2'])
print("Shape of matrix after one hot encoding ",tfidf_test_ques2.shape)
print("the number of unique words ", tfidf_test_ques2.get_shape()[1])
```

```
↳ Shape of matrix after one hot encoding (12500, 13369)
   the number of unique words 13369
   Shape of matrix after one hot encoding (12500, 13369)
   the number of unique words 13369
```

```
# extraction features from test data frame
X_test_feature_df = X_test.drop(['id','qid1','qid2','question1','question2'], axis=1, inplace=False)
```

```
print("test Shape Before -> ",X_test_feature_df.shape," Type",type(X_test_feature_df))
```

```
#so we need to convert our feature data into sparse matrix so that we will combine our feature and a
test_feat_sparse = scipy.sparse.csr_matrix(X_test_feature_df)
```

```
print("test Shape After-> ",test_feat_sparse.shape," Type",type(test_feat_sparse))
```

```
↳ test Shape Before -> (12500, 26) Type <class 'pandas.core.frame.DataFrame'>
   test Shape After-> (12500, 26) Type <class 'scipy.sparse.csr.csr_matrix'>
```

```
# combining our tfidf and features into one
```

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
```

```
tfidf_train = hstack((tfidf_train_ques1,tfidf_train_ques2))
```

```
# test features(feat + tfidfvec)
tfidf_test = hstack((tfidf_test_ques1,tfidf_test_ques2))
```

```
#final train and test data shape
print("train data shape",tfidf_train.shape)
```

```
print("Test data shape ",tfidf_test.shape)
```

```
↳ train data shape (37500, 26738)
   Test data shape (12500, 26738)
```

```
tfidf_train.shape
```

```
↳ (37500, 26738)
```

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

```
tfidf_train = hstack((train_feat_sparse,tfidf_train_ques1,tfidf_train_ques2))
```

```
# test features(feat + tfidfvec)
tfidf_test = hstack((test_feat_sparse,tfidf_test_ques1,tfidf_test_ques2))
```

```
#final train and test data shape
```

```
print("train data shape",tfidf_train.shape)
```

```
print("Test data shape ",tfidf_test.shape)
```

```
↳ train data shape (37500, 26764)
   Test data shape (12500, 26764)
```

```
print("Final Shape of the Data matrix")
print(tfidf_train.shape, y_train.shape)
```

```
print(tfidf_test.shape, y_test.shape)
```

```
↳ Final Shape of the Data matrix
   (37500, 26764) (37500,)
   (12500, 26764) (12500,)
```

```
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 test data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[0])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
↳ ----- Distribution of output variable in train data -----
   Class 0:  0.6270133333333333 Class 1:  0.3729866666666667
   ----- Distribution of output variable in test data -----
   Class 0:  0.62704 Class 1:  0.37296
```

▼ 5.4 Logistic Regression with hyperparameter tuning

```
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\_model.SGDClassifier.html
# -----
```

```
# default parameters
```

```
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal, eta0=0.01,
# class_weight=None, warm_start=False, average=False, n_iter=None)
```

```
# some of methods
```

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

```
log_error_array=[]
```

```
for i in alpha:
```

```
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
```

```
    clf.fit(tfidf_train, y_train)
```

```
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
    sig_clf.fit(tfidf_train, y_train)
```

```
    predict_y = sig_clf.predict_proba(tfidf_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.classes_, eps=1e-15))
```

```

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(tfidf_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(tfidf_train, y_train)

predict_y = sig_clf.predict_proba(tfidf_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(tfidf_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
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: 0.43243610251165965
For values of alpha = 0.0001 The log loss is: 0.4420792846640858
For values of alpha = 0.001 The log loss is: 0.4427199998389551
For values of alpha = 0.01 The log loss is: 0.4711340724303233
For values of alpha = 0.1 The log loss is: 0.47961386720378796
For values of alpha = 1 The log loss is: 0.5096511886344345

```

5.5 Linear SVM with hyperparameter tuning



```
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\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.1,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

```

```

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(tfidf_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(tfidf_train, y_train)
    predict_y = sig_clf.predict_proba(tfidf_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.classes_))

```

```

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)
clf.fit(tfidf_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(tfidf_train, y_train)

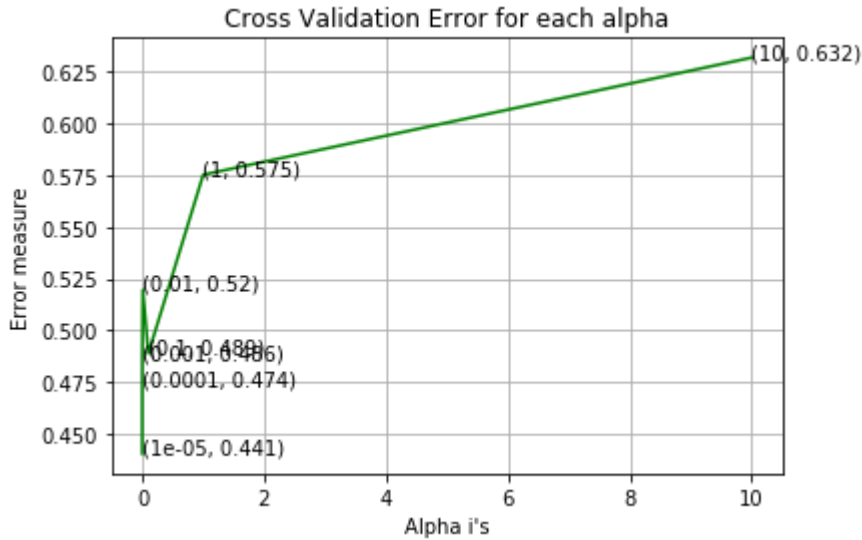
predict_y = sig_clf.predict_proba(tfidf_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(tfidf_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
predicted_y = np.argmax(predict_y, axis=1)
print("Total number of data points :", len(predicted_y))

```

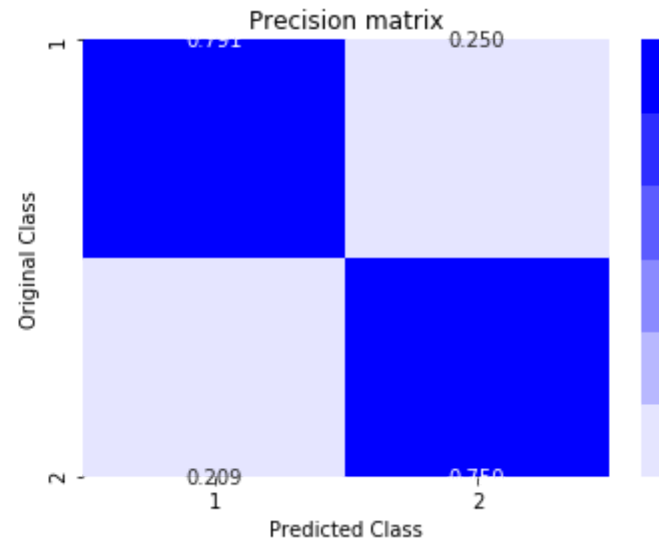
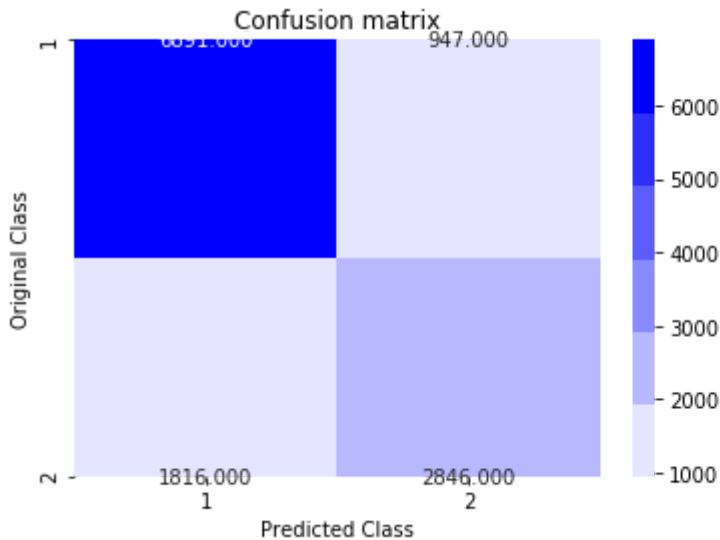
plot_confusion_matrix(y_test, predicted_y)



For values of alpha = 1e-05 The log loss is: 0.44055060308003713
 For values of alpha = 0.0001 The log loss is: 0.4739964863198457
 For values of alpha = 0.001 The log loss is: 0.4861788835774691
 For values of alpha = 0.01 The log loss is: 0.5196767681537059
 For values of alpha = 0.1 The log loss is: 0.48887010386648644
 For values of alpha = 1 The log loss is: 0.5751288906659099
 For values of alpha = 10 The log loss is: 0.6316597464583033



For values of best alpha = 1e-05 The train log loss is: 0.42745588054917155
 For values of best alpha = 1e-05 The test log loss is: 0.44055060308003713
 Total number of data points : 12500



5.6 XGBoost

▾ A. Hyperparameter Tuning

```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
import scipy.stats as sc
```

```
params = {

    "learning_rate":sc.uniform(0.05,0.3),
```

```

        'max_depth': sc.randint(3,15),
        'n_estimators' : sc.randint(10,200),
        "min_child_weight" : [ 1, 3, 5, 7 ],
        'gamma': sc.uniform(0.0,0.5)
    }
x_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss',n_jobs=-1)

xgb_random_search = RandomizedSearchCV(x_model, param_distributions = params,n_iter=30,
                                       scoring = 'neg_log_loss', n_jobs = -1,cv=3)

xgb_random_search.fit(tfidf_train, y_train)

print("Score :",xgb_random_search.best_score_)
print("Best Params",xgb_random_search.best_params_)

```

```

➡ Score : -0.3497625287179327
   Best Params {'gamma': 0.12526728798377457, 'learning_rate': 0.08528144281663791, 'max_depth':

```

▼ B. With Best Params

```

bst = xgb.XGBClassifier(max_depth=12,learning_rate= 0.08528144281663791,objective='binary:logistic')
bst.fit(tfidf_train, y_train)

clf_calib = CalibratedClassifierCV(bst, method="sigmoid")
clf_calib.fit(tfidf_train, y_train)

predict_y = clf_calib.predict_proba(tfidf_train)

print("The train log loss is: ",log_loss(y_train, predict_y,labels=bst.classes_, eps=1e-15))

predict_y = clf_calib.predict_proba(tfidf_test)
print("The test log loss is : ",log_loss(y_test, predict_y,labels=bst.classes_, eps=1e-15))

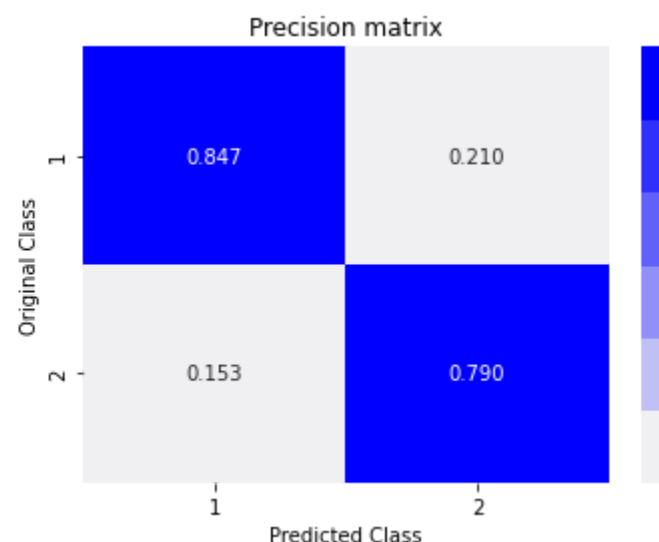
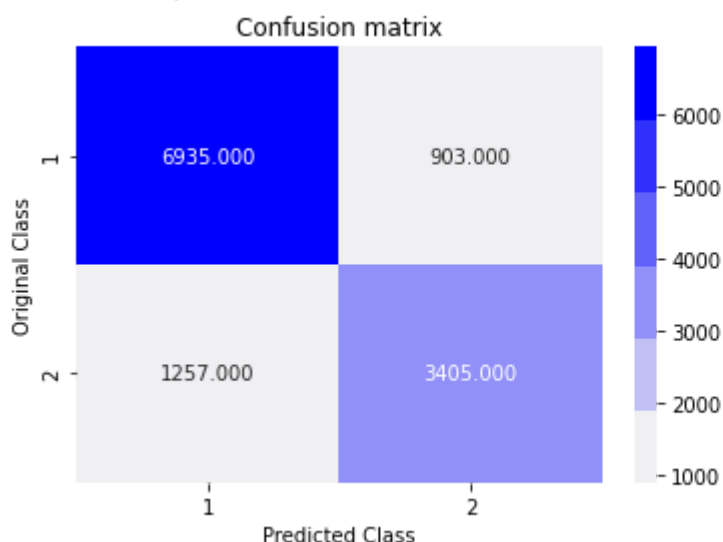
predicted_y =np.argmax(predict_y,axis=1)
plot_confusion_matrix(y_test, predicted_y)

```

```


➡ The train log loss is: 0.2483836581900135
   The test log loss is : 0.3533921714478344

```



Conclusions And Observations:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ['Serial No.', 'Model Name', 'Tokenizer', 'Hyperparameter Tunning', 'Test Log Lo
ptable.add_row(["1","Random","TFIDF Weighted W2V","-", "0.89"])
ptable.add_row(["2","Logistic Regression","TFIDF Weighted W2V","Done", "0.468"])
ptable.add_row(["3","Linear SVM","TFIDF Weighted W2V","Done", "0.474"])
ptable.add_row(["4","XGBoost","TFIDF Weighted W2V","-", "0.399"])
ptable.add_row(["\n", "\n", "\n", "\n", "\n"])
ptable.add_row(["1","Random","TFIDF", "- ", "0.89"])
ptable.add_row(["2","Logistic Regression","TFIDF", "Done", "0.432"])
ptable.add_row(["3","Linear SVM","TFIDF", "Done", "0.440"])
ptable.add_row(["4","XGBoost","TFIDF", "Done", "0.353"])
print(ptable)
```



Serial No.	Model Name	Tokenizer	Hyperparameter Tunning	Test Log Lo
1	Random	TFIDF Weighted W2V	-	0.89
2	Logistic Regression	TFIDF Weighted W2V	Done	0.468
3	Linear SVM	TFIDF Weighted W2V	Done	0.474
4	XGBoost	TFIDF Weighted W2V	-	0.399
1	Random	TFIDF	-	0.89
2	Logistic Regression	TFIDF	Done	0.432
3	Linear SVM	TFIDF	Done	0.440
4	XGBoost	TFIDF	Done	0.353

STEP BY STEP PROCEDURE:

- 1.Our Data contains 5 columns : qid1, qid2, question1, question2, is_duplicate from which 'is_duplicate' is a class label which specifies whether the question 1 and question 2 are similar or not and this is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.
- 2.Firstly we preprocessed our data,did feature engineering to create new features which might help us and created our dataframes, then we merged dataframes and got out final matrix.Now after doing simple EDA on dataset we will try some Basic Feature Extraction (before cleaning) the dataset like Frequency of qid1's ,word_Common and etc. and using this featured dataset we will do some EDA on it so that we will able to rectify which features are most useful features out of all features i.e(whic feature is helpful for classification)
- 3.After doing basic Basic feature extractions we will try some Advanced Feature Extraction using NLP and Fuzzy Features but before doing this we will do Preprocessing of Text and then we will do Advanced Feature Extraction and try to visualise our Advanced Feature using EDA, PCA and word clouds.
- 4.Then we Split the data randomly . We could also have done time based splitting, since the model could predict for future unseen data too. But, there was no timestamp column provided, so the only option was

to split it randomly.

5. Now as we know we have columns of two questions i.e question 1 and question 2 and we will vectorize that both col using tfidf weighted word-vectors so that we will be able to apply models on it and after doing all these we will merge all the features i.e basic features + advance features + question1 tfidf w2v + and question 2 tfidf w2v. and Now after doing all of these we will apply models on it.

6. Here as we know here we are using two main performance matrix in this case study i.e log-loss and confusion matrix and using these we will get our performance of the models

7. Let's start: here we are there model i.e Logistic Regression linear svm and XgBoost and a random model which Finding worst-case log-loss and then we try to compare all

8. In next step we will try our models with other vectorizer i.e tfidf instead of tfidf weighted w2v and try to do some hyperparameter tuning in order to improve the model performance.

9. Now, we have applied simple Random/Dumb Model. It gave a log loss of 0.89. This is the worst case log-loss. This will act as a base and any model we design should have a log-loss lesser than this dumb model.

10. After that we have applied Logistic Regression with hyperparameter tuning. It gave a log-loss of 0.43, which is lower than Random Model. We can also see that there is no Overfitting problem, since, Train log-loss and Test log-loss are very close.

11. After that we have applied Linear SVM with hyperparameter tuning. It gave the log-loss of 0.44, which is lower than Random Model. We can also see that there is no Overfitting problem, since, Train log-loss and Test log-loss are very close.

12. After that we have applied Xgboost with hyperparameter tuning. It gave the log-loss of 0.34, which is lower than Random Model. We can also see that there is no Overfitting problem, since, Train log-loss and Test log-loss are very close.

Looks like among all the models that we tried Xgboost seems to perform well and hence can be used to Identify which questions asked on Quora are duplicates of questions that have already been asked.

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.