

### 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
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
```



```
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")
from datetime 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
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")

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

```

 Number of data points: 404290

```
df.head()
```

	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	<b>question2</b>
<b>0</b>	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guid
<b>1</b>	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian
<b>2</b>	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increa
<b>3</b>	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]
<b>4</b>	4	9	10	Which one dissolve in water quickly sugar, salt...	Which fish would sur

```
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

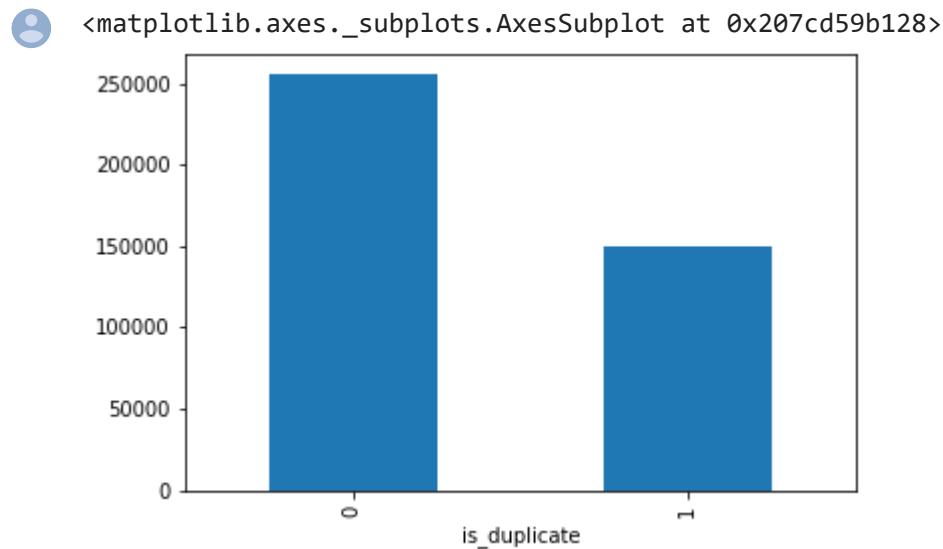
We are given a minimal number of data fields here, consisting of:

- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is\_duplicate: The label that we are trying to predict - whether the two questions are duplicates or not

### 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()
```



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

~> Total number of question pairs for training:  
404290

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

~> Question pairs are not Similar (is\_duplicate = 0):  
63.08%

~> Question pairs are Similar (is\_duplicate = 1):  
36.92%

### 3.2.2 Number of unique questions

```
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*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
```

~> Total number of Unique Questions are: 537933

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

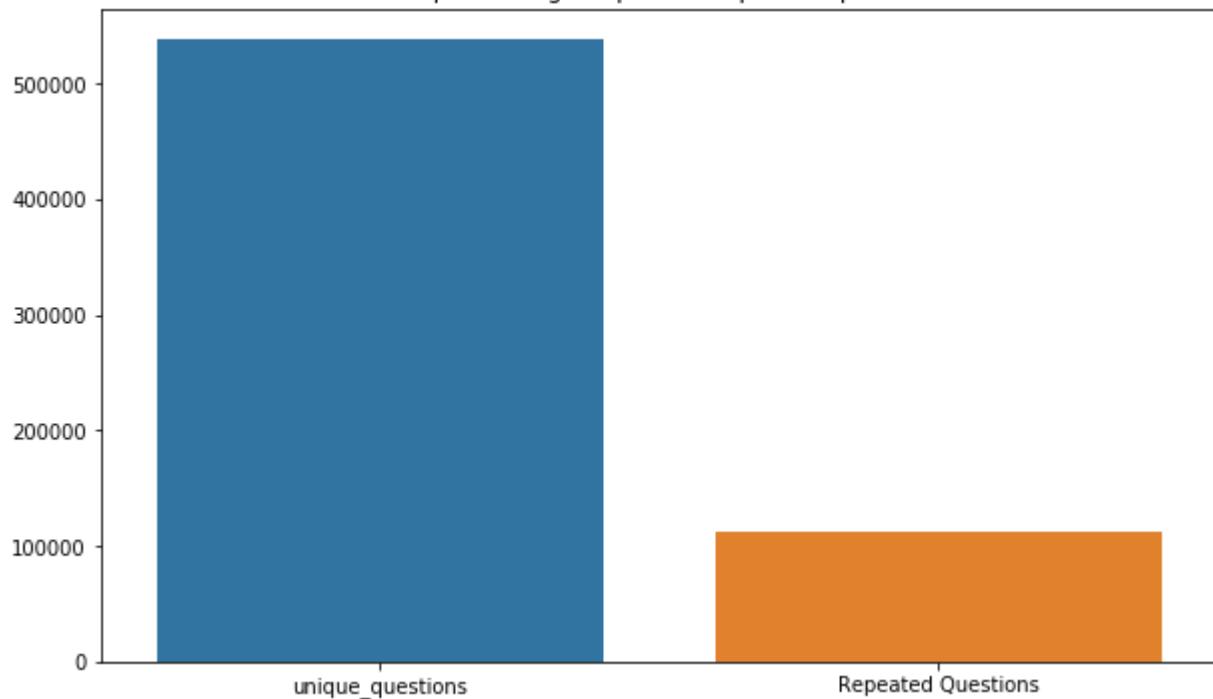
Max number of times a single question is repeated: 157

```
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()
```



Plot representing unique and repeated questions



### 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_i

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

➊ 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')

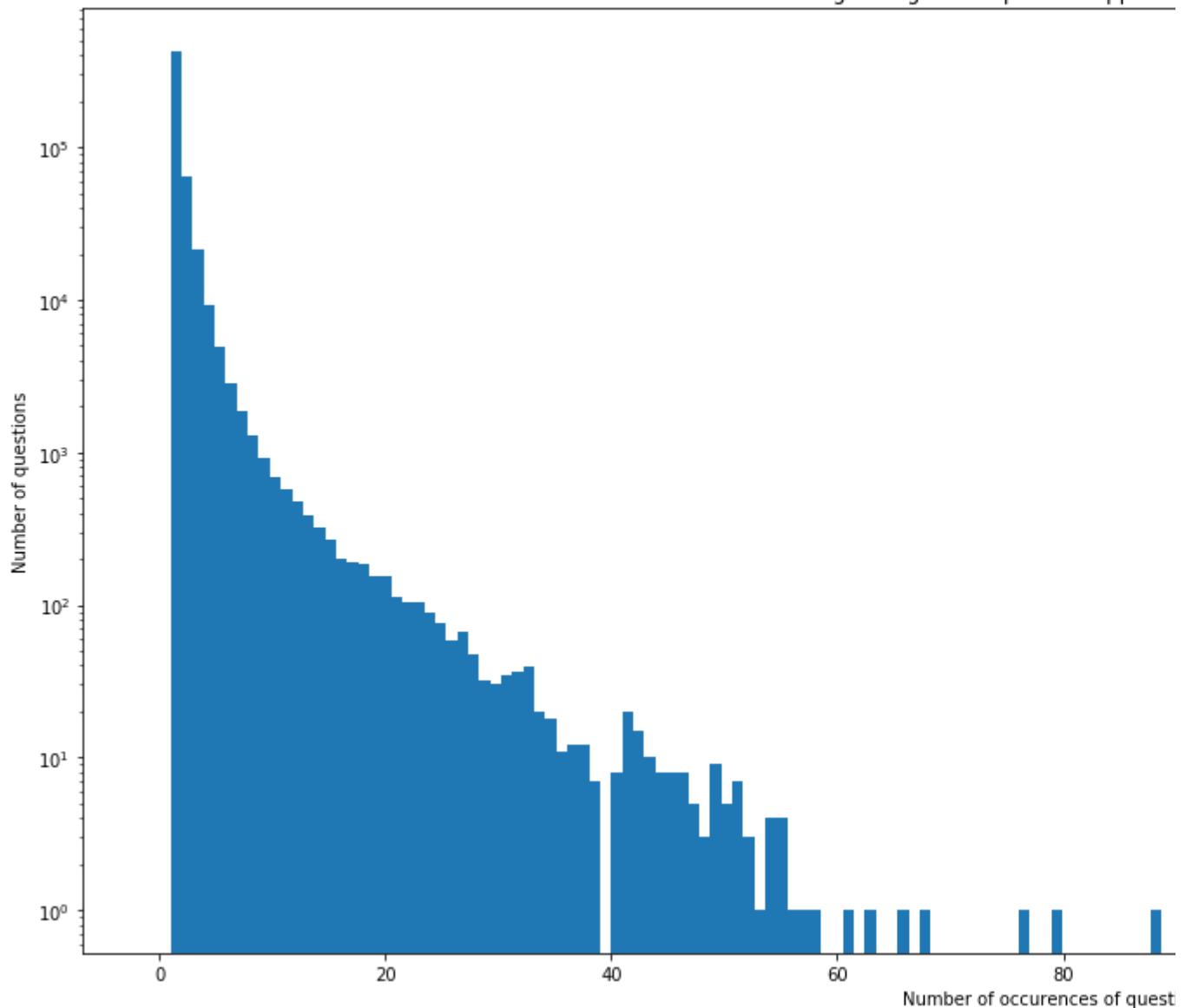
plt.ylabel('Number of questions')

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



Maximum number of times a single question is repeated: 157

Log-Histogram of question appeara



### 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      qid2           question1  \
• There are two rows with null values in question2
      363362  363362  493340  493341           NaN

# 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()
```



### 3.3.1 Analysis of some of the extracted features

What is the

- Here are some questions have only one single words.

to invest in your w

```
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])
```

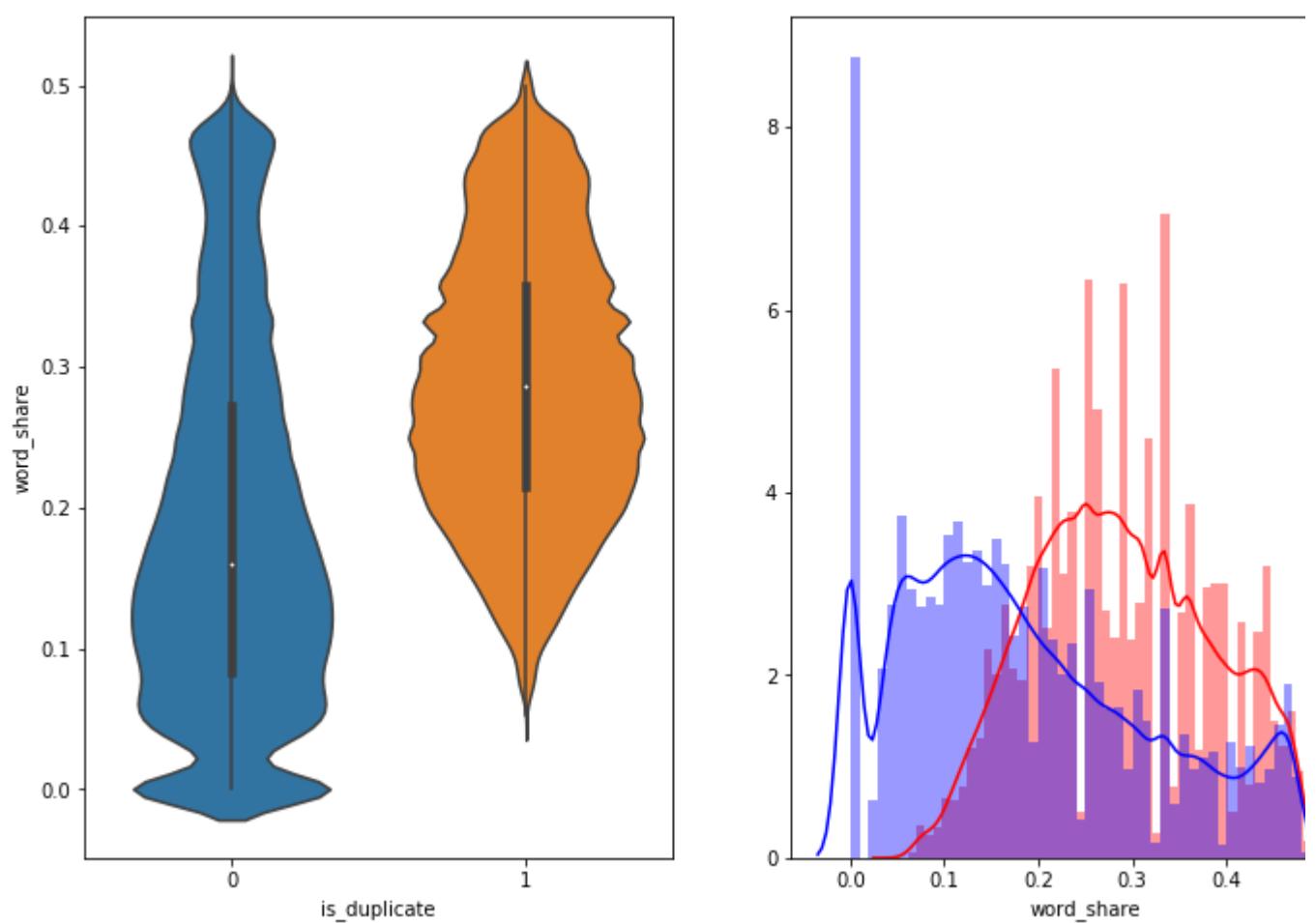
👤 Minimum length of the questions in question1 : 1  
Minimum length of the questions in question2 : 1  
Number of Questions with minimum length [question1] : 67  
Number of Questions with minimum length [question2] : 24  
... . increased by

#### 3.3.1.1 Feature: word\_share

very am I find the

```
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., t word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are dupl

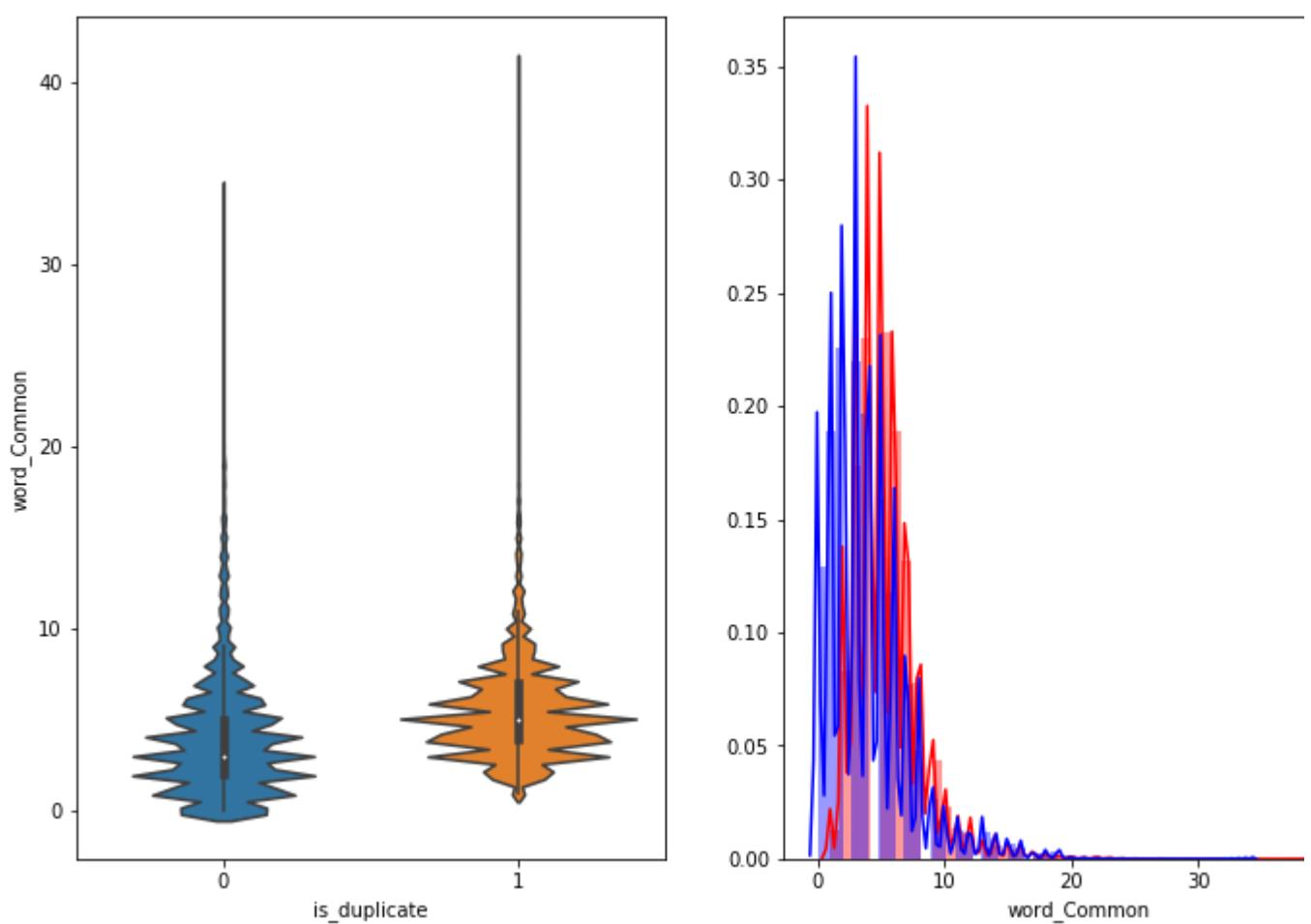
### 3.3.1.2 Feature: word\_Common

```
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 overlap

## ▼ 3.4 Preprocessing of Text

```
# To get the results in 4 decimal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")

def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("'", "").replace("''", "")\
        .replace("won't", "will not").replace("cannot", "can not").replace\
        ("n't", " not").replace("what's", "what is").replace("it's\
        .replace("ve", " have").replace("i'm", "i am").replace("re", " a\
        .replace("he's", "he is").replace("she's", "she is").replace("s",\
        .replace("%", " percent ").replace("₹", " rupee ").replace("$", "\
        .replace("€", " euro ").replace("ll", " will")

    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)
```

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

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

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

return x
```

## ▼ 3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

```
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.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
# https://github.com/seatgeek/fuzzywuzzy
print("fuzzy features..")

df["token_set_ratio"]      = df.apply(lambda x: fuzz.token_set_ratio(x["question1"], x["question2"]))
# The token sort approach involves tokenizing the string in question, sorting the tokens
# then joining them back into a string. We then compare the transformed strings with a sim
df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"], x["question2"]))
df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]))
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["question2"]))

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)
```

Extracting features for train:  
token features...  
fuzzy features..

	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	<b>question2</b>	<b>is_duplicate</b>	<b>cwc_min</b>	<b>cwc_max</b>	<b>csc_min</b>	<b>csc_max</b>
<b>0</b>	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
<b>1</b>	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	0.599981

2 rows × 21 columns

## ▼ 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.vstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
n = np.vstack([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()

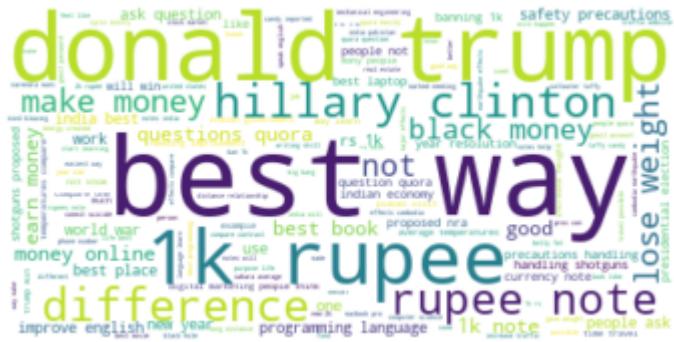
stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

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 : 16110077  
Total number of words in non duplicate pair questions : 33193603

```
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()
```

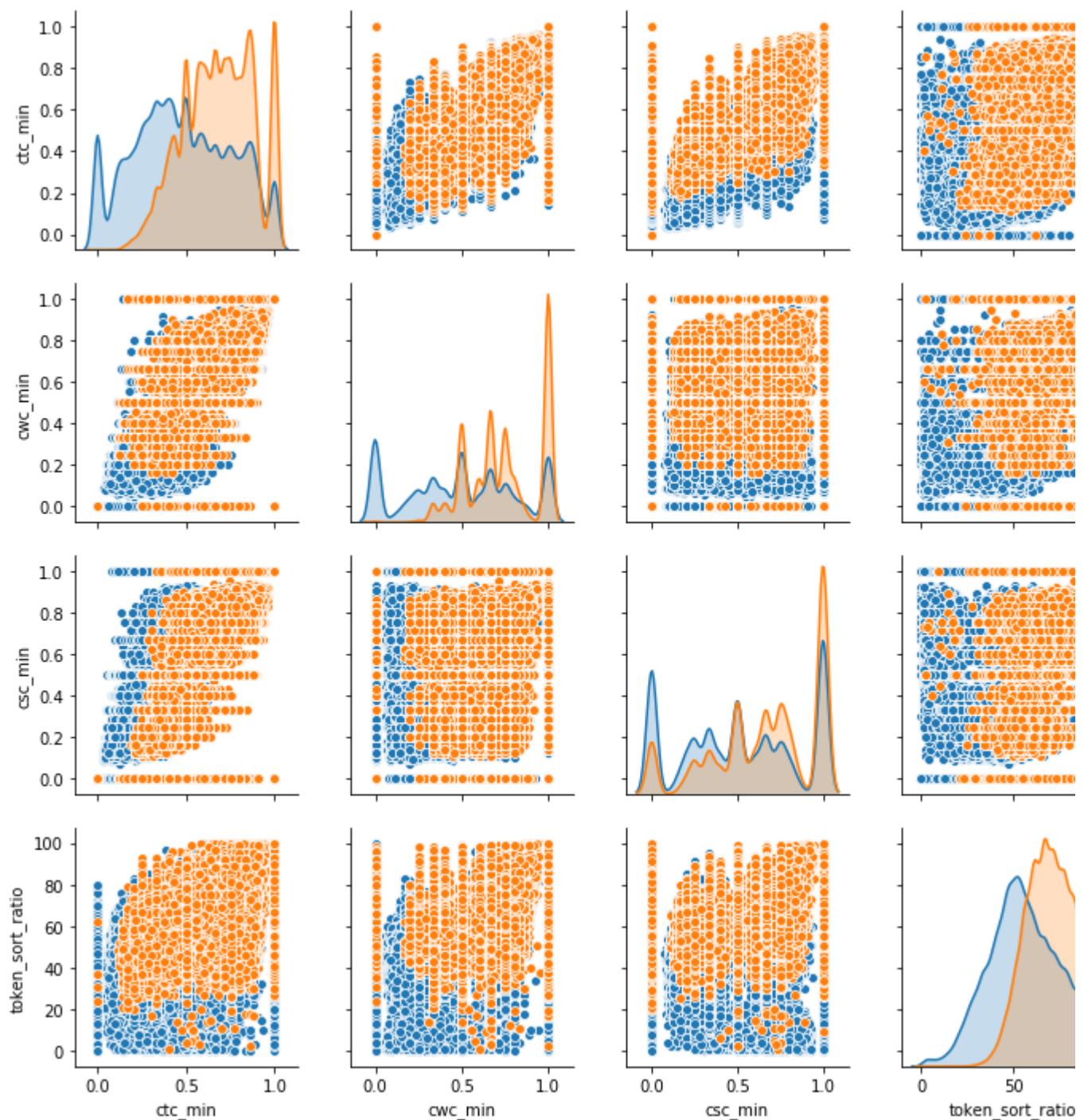
### Word Cloud for non-Duplicate Question pairs:



### 3.5.1.2 Pair plot of features ['ctc\_min', 'cwc\_min', 'csc\_min', 'tok\_min', 'tok\_max', 'tok\_mean', 'tok\_std', 'tok\_var']

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

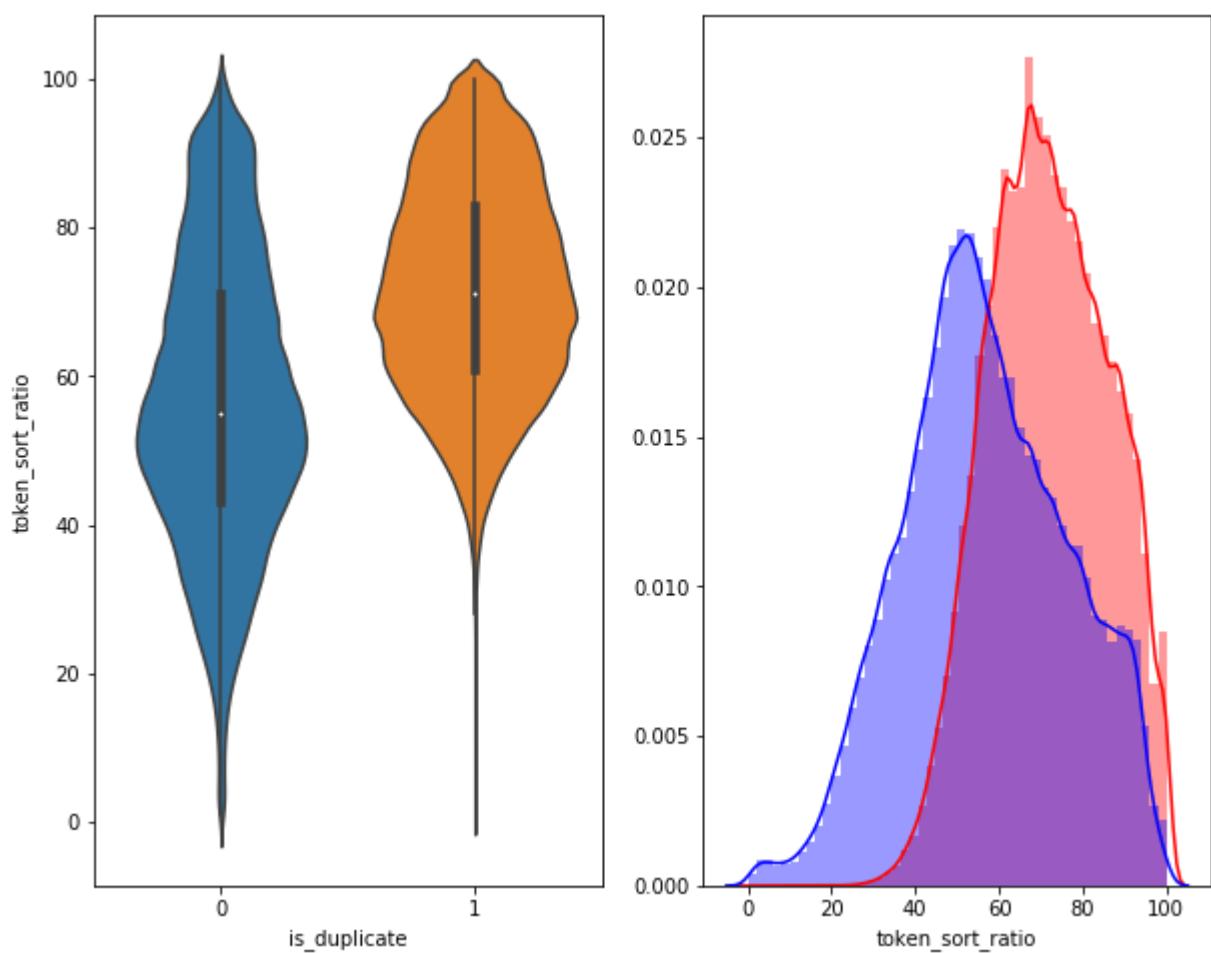




```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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

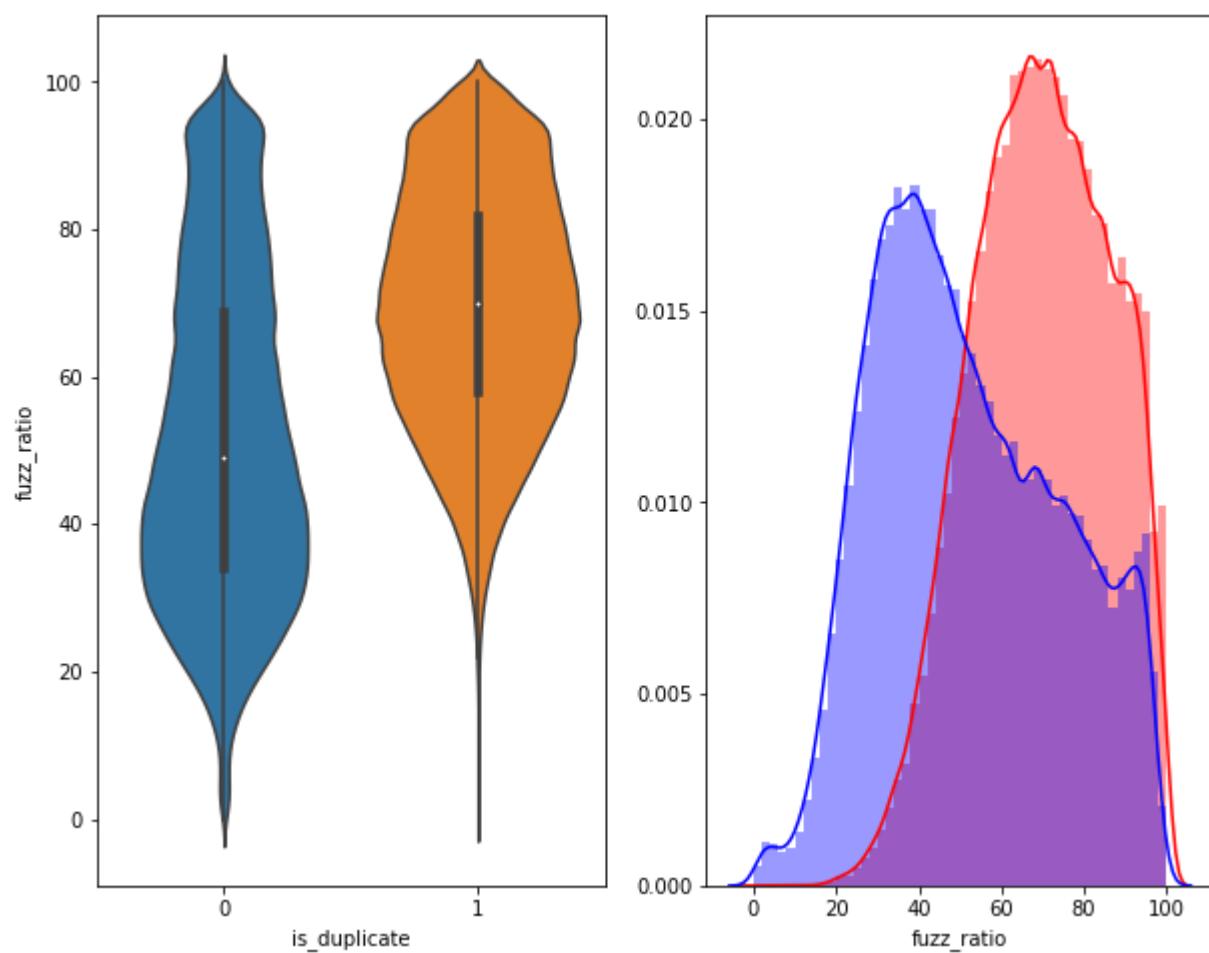
plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['token_sort_ratio'][0:] , label = "1", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" , color = 'blue')
plt.show()
```



```
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)

from sklearn.preprocessing import MinMaxScaler

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

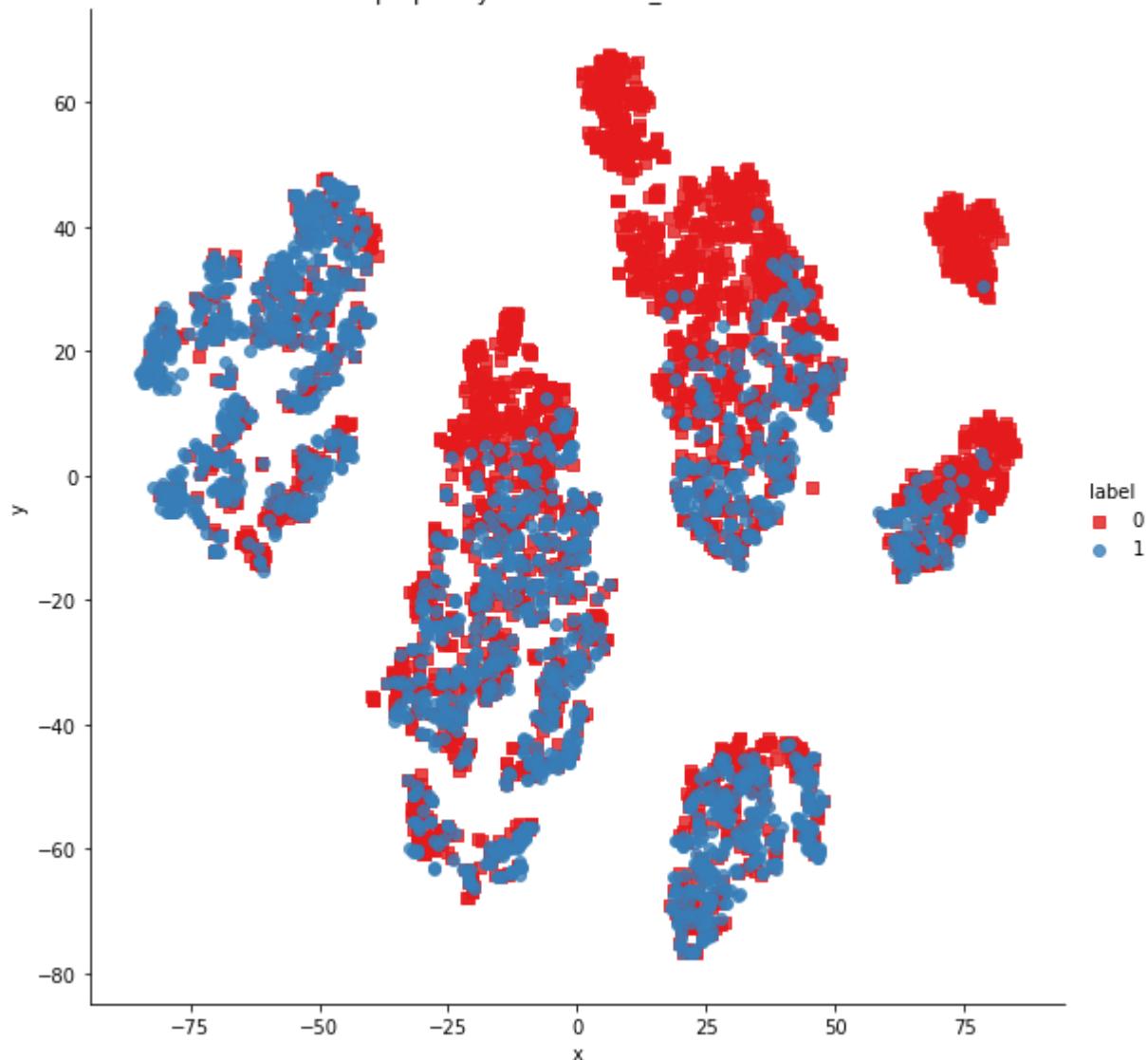
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.018s...
[t-SNE] Computed neighbors for 5000 samples in 0.778s...
[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: 0.131928
[t-SNE] Computed conditional probabilities in 0.444s
[t-SNE] Iteration 50: error = 81.2975616, gradient norm = 0.0496455 (50 iterations in 6.
[t-SNE] Iteration 100: error = 70.6435165, gradient norm = 0.0094614 (50 iterations in 4
[t-SNE] Iteration 150: error = 68.9952850, gradient norm = 0.0056374 (50 iterations in 4
[t-SNE] Iteration 200: error = 68.2175980, gradient norm = 0.0044332 (50 iterations in 4
[t-SNE] Iteration 250: error = 67.7385254, gradient norm = 0.0034321 (50 iterations in 4
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.738525
[t-SNE] Iteration 300: error = 1.7930490, gradient norm = 0.0011818 (50 iterations in 4.
[t-SNE] Iteration 350: error = 1.3966638, gradient norm = 0.0004836 (50 iterations in 4.
[t-SNE] Iteration 400: error = 1.2328721, gradient norm = 0.0002750 (50 iterations in 4.
[t-SNE] Iteration 450: error = 1.1440563, gradient norm = 0.0001877 (50 iterations in 4.
[t-SNE] Iteration 500: error = 1.0895753, gradient norm = 0.0001404 (50 iterations in 4.
[t-SNE] Iteration 550: error = 1.0542322, gradient norm = 0.0001145 (50 iterations in 4.
[t-SNE] Iteration 600: error = 1.0302582, gradient norm = 0.0001017 (50 iterations in 4.
[t-SNE] Iteration 650: error = 1.0142238, gradient norm = 0.0000900 (50 iterations in 4.
[t-SNE] Iteration 700: error = 1.0029600, gradient norm = 0.0000806 (50 iterations in 4.
[t-SNE] Iteration 750: error = 0.9942252, gradient norm = 0.0000781 (50 iterations in 4.
[t-SNE] Iteration 800: error = 0.9875125, gradient norm = 0.0000736 (50 iterations in 4.
[t-SNE] Iteration 850: error = 0.9824185, gradient norm = 0.0000673 (50 iterations in 4.
[t-SNE] Iteration 900: error = 0.9780059, gradient norm = 0.0000659 (50 iterations in 4.
[t-SNE] Iteration 950: error = 0.9744161, gradient norm = 0.0000617 (50 iterations in 4.
[t-SNE] Iteration 1000: error = 0.9713724, gradient norm = 0.0000583 (50 iterations in 4
[t-SNE] KL divergence after 1000 iterations: 0.971372
```

```
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=[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 0.021s...
[t-SNE] Computed neighbors for 5000 samples in 0.779s...
[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: 0.131928
[t-SNE] Computed conditional probabilities in 0.399s
[t-SNE] Iteration 50: error = 80.5249557, gradient norm = 0.0319611 (50 iterations in 20
[t-SNE] Iteration 100: error = 69.4158859, gradient norm = 0.0033386 (50 iterations in 1
[t-SNE] Iteration 150: error = 68.0448608, gradient norm = 0.0019634 (50 iterations in 9
[t-SNE] Iteration 200: error = 67.4930801, gradient norm = 0.0011609 (50 iterations in 1
[t-SNE] Iteration 250: error = 67.1813202, gradient norm = 0.0008686 (50 iterations in 9
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.181320
[t-SNE] Iteration 300: error = 1.5266187, gradient norm = 0.0007106 (50 iterations in 12
[t-SNE] Iteration 350: error = 1.1894693, gradient norm = 0.0001963 (50 iterations in 16
[t-SNE] Iteration 400: error = 1.0453291, gradient norm = 0.0000995 (50 iterations in 16
[t-SNE] Iteration 450: error = 0.9735472, gradient norm = 0.0000740 (50 iterations in 15
[t-SNE] Iteration 500: error = 0.9391118, gradient norm = 0.0000586 (50 iterations in 15
[t-SNE] Iteration 550: error = 0.9216439, gradient norm = 0.0000491 (50 iterations in 15
[t-SNE] Iteration 600: error = 0.9106981, gradient norm = 0.0000487 (50 iterations in 15
[t-SNE] Iteration 650: error = 0.9030094, gradient norm = 0.0000377 (50 iterations in 15
[t-SNE] Iteration 700: error = 0.8947795, gradient norm = 0.0000328 (50 iterations in 16
[t-SNE] Iteration 750: error = 0.8864105, gradient norm = 0.0000338 (50 iterations in 16
[t-SNE] Iteration 800: error = 0.8798748, gradient norm = 0.0000314 (50 iterations in 15
[t-SNE] Iteration 850: error = 0.8745480, gradient norm = 0.0000292 (50 iterations in 15
[t-SNE] Iteration 900: error = 0.8701542, gradient norm = 0.0000287 (50 iterations in 15
[t-SNE] Iteration 950: error = 0.8666047, gradient norm = 0.0000262 (50 iterations in 15
[t-SNE] Iteration 1000: error = 0.8636045, gradient norm = 0.0000248 (50 iterations in 1
[t-SNE] KL divergence after 1000 iterations: 0.863604
```

```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

data=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered features')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```



- ▼ Featurizing text data with tfidf weighted word-vectors

```
import pandas as pd
import matplotlib.pyplot as plt
import re
```

<https://colab.research.google.com/drive/1bbr-plsv3udKh5UzXVF53bNL3jp3Czn-#scrollTo=4BJPzOpfxB5F&printMode=true>

```

import time
import warnings
import numpy as npx
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()
```

		<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	
<b>0</b>	0	1	2		What is the step by step guide to invest in sh...	What is the step by step guid
<b>1</b>	1	3	4		What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian
<b>2</b>	2	5	6		How can I increase the speed of my internet co...	How can Internet speed be incre
<b>3</b>	3	7	8		Why am I mentally very lonely? How can I solve...	Find the remainder when [math]
<b>4</b>	4	9	10		Which one dissolve in water quickly sugar, salt...	Which fish would sur

```

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))

```

After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by GLOVE model which comes free with "Spacy". <https://spacy.io/usage/vectors-similarity> It is trained on terms of word semantics.

```
# 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% |  | 404290/40

```
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)
```

100% | 404290/4

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

	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	
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 go...

```
# data before preprocessing
dfppro.head(2)
```

	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	<b>question2</b>	<b>is_duplicate</b>	<b>freq_qid1</b>	<b>freq_qid2</b>	<b>q1len</b>	<b>q2len</b>
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	4	1	51	8

```
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
0	211.129864	-144.683059	-68.811247	-153.662141	-89.931593	2.311301	136.743747
1	144.124685	-114.012484	-111.716694	-104.885038	-88.238478	16.441834	58.238013
2	81.757898	-142.184507	0.559867	-104.660084	-84.156631	22.515110	115.521661
3	-126.651922	-59.747160	-67.763201	-138.114731	-101.038699	88.148523	-22.912261
4	299.444044	-188.632001	-22.946291	-273.683355	-188.480395	107.123044	174.946302

5 rows × 96 columns

```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

	0	1	2	3	4	5	6
0	151.268526	-127.013168	-31.546286	-142.905807	-97.249094	9.485758	106.682259
1	152.023095	-44.955390	-103.559249	-128.467601	-118.567610	44.577916	137.906144
2	4.930220	-29.029581	-117.808812	-98.332275	-19.064096	-9.867805	141.808202
3	-6.951929	-44.951731	-17.343082	-61.444452	-7.469152	16.942014	95.049250
4	96.174524	-71.613948	21.584882	-92.742468	-106.643129	10.646790	92.190157

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	
0	0	0	0	1	1	66	57	14	
1	1	1	0	4	1	51	88	8	
2	2	2	0	1	1	73	59	14	

3 rows × 47 columns

### 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['
```

```
X_train.shape
```

👤 (37500, 47)

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

👤

	Unnamed: 0	id	is_duplicate	freq_qid1_x	freq_qid2_x	q1len_x	q2len_x	q1_n_w
23561	23561	23561	0	1	1	33	50	
3536	3536	3536	0	1	1	46	58	

2 rows × 47 columns

```

# 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)

❷ Number of data points in train data : (37500, 44)
Number of data points in test data : (12500, 44)

y_train.shape

❷ (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 train 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)

❷ ----- Distribution of output variable in train data -----
Class 0: 0.6270133333333333 Class 1: 0.3729866666666667
----- Distribution of output variable in train data -----
Class 0: 0.37296 Class 1: 0.37296

# 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 cl

    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 dia
    # 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 dia
# 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)

```

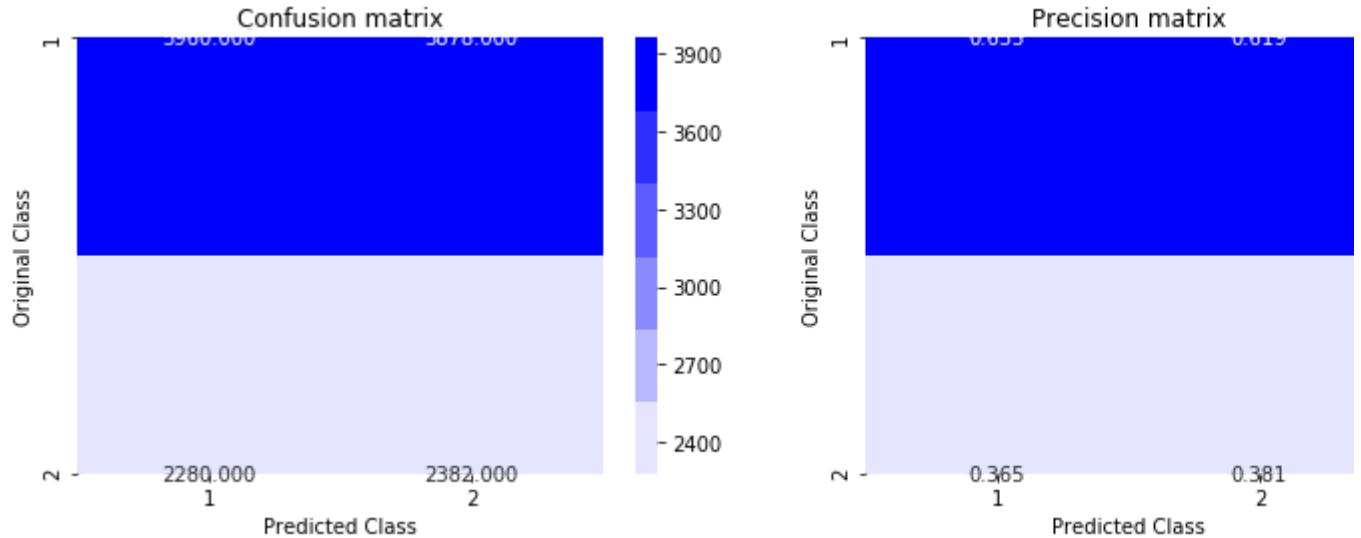
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
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 0.8746716989316823



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

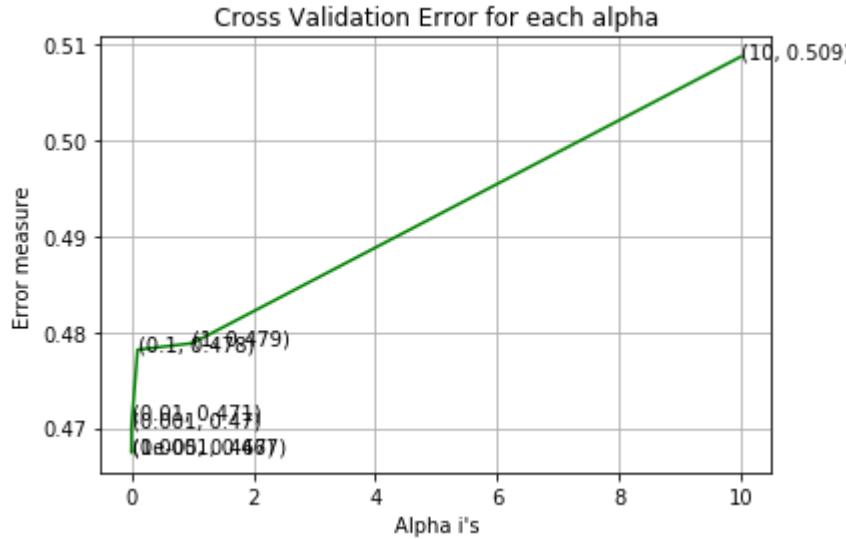
```
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, 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_t  
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_te  
predicted_y =np.argmax(predict_y, axis=1)  
print("Total number of data points :", len(predicted_y))  
plot_confusion_matrix(y_test, predicted_y)
```



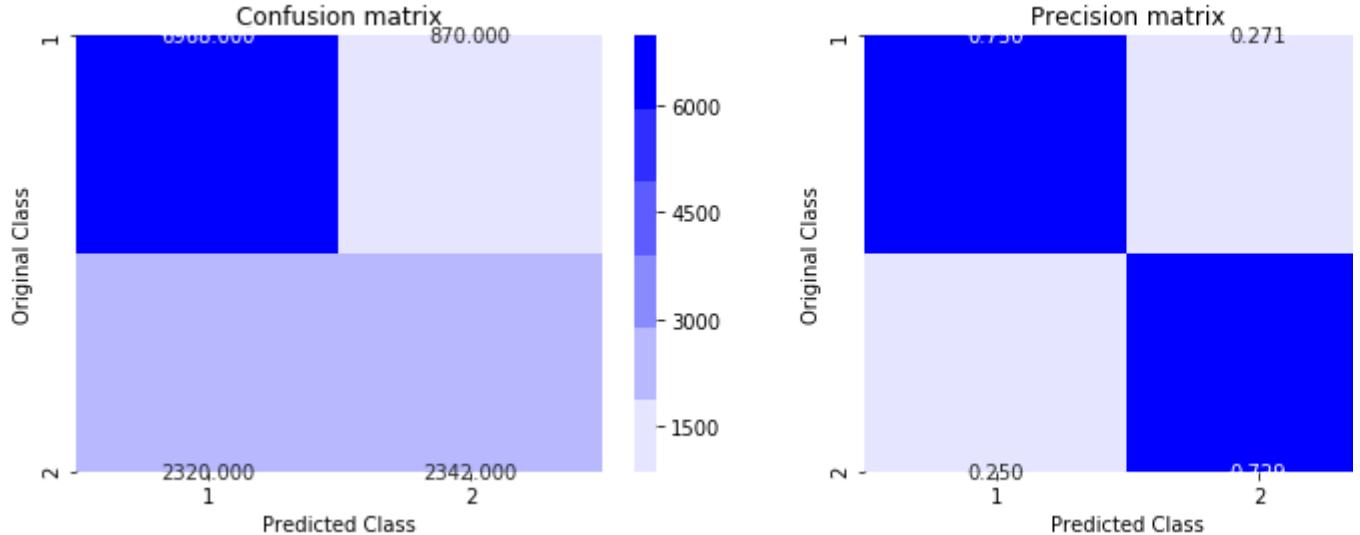
```
For values of alpha = 1e-05 The log loss is: 0.46748950473787415
For values of alpha = 0.0001 The log loss is: 0.467411411795257
For values of alpha = 0.001 The log loss is: 0.4702043503620991
For values of alpha = 0.01 The log loss is: 0.47095651464973903
For values of alpha = 0.1 The log loss is: 0.47815931432174075
For values of alpha = 1 The log loss is: 0.47883932764652315
For values of alpha = 10 The log loss is: 0.5087726243467064
```



For values of best alpha = 0.0001 The train log loss is: 0.4658529781704176

For values of best alpha = 0.0001 The test log loss is: 0.467411411795257

Total number of data points : 12500



## 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
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
```

```
# n_jobs=1, verbose=0, epsilon=0.1, n_iter=None, learning_rate='optimal',
# 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_))

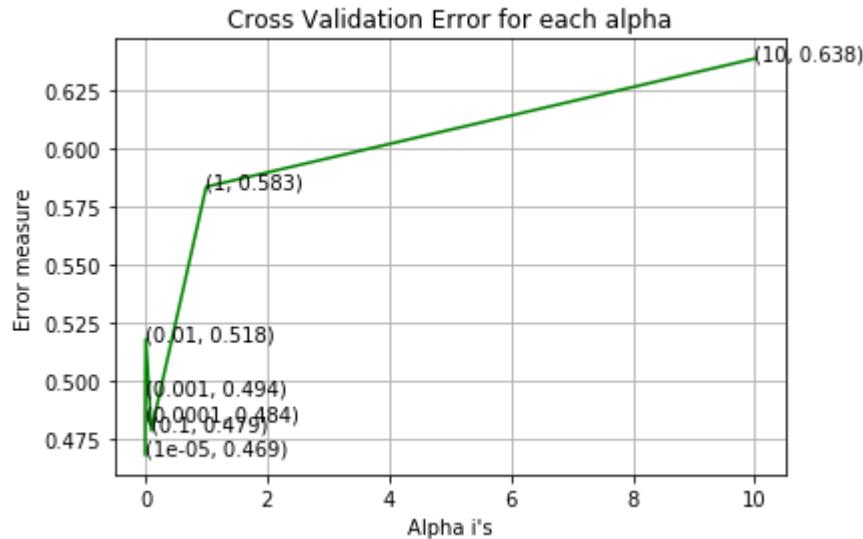
fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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



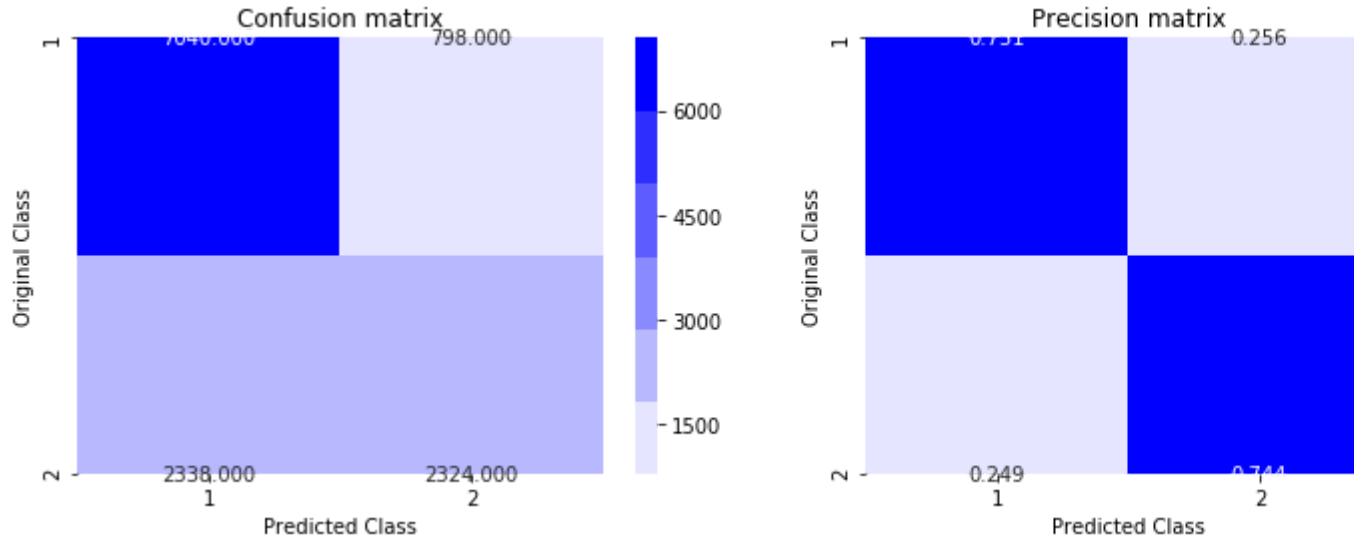
```
For values of alpha = 1e-05 The log loss is: 0.468568715140383
For values of alpha = 0.0001 The log loss is: 0.4836764509430551
For values of alpha = 0.001 The log loss is: 0.49419792841068927
For values of alpha = 0.01 The log loss is: 0.5181667966968087
For values of alpha = 0.1 The log loss is: 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')]

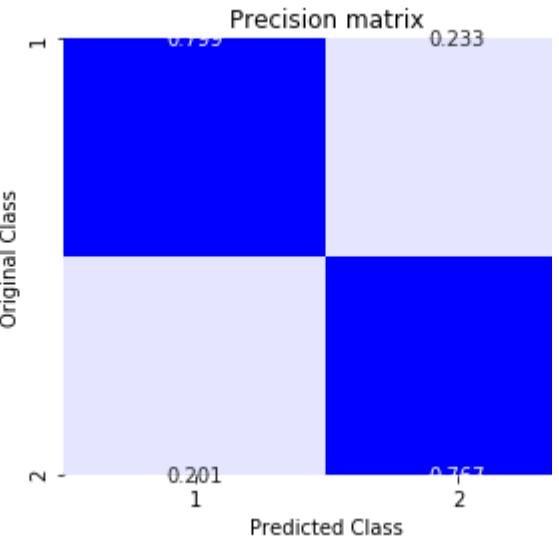
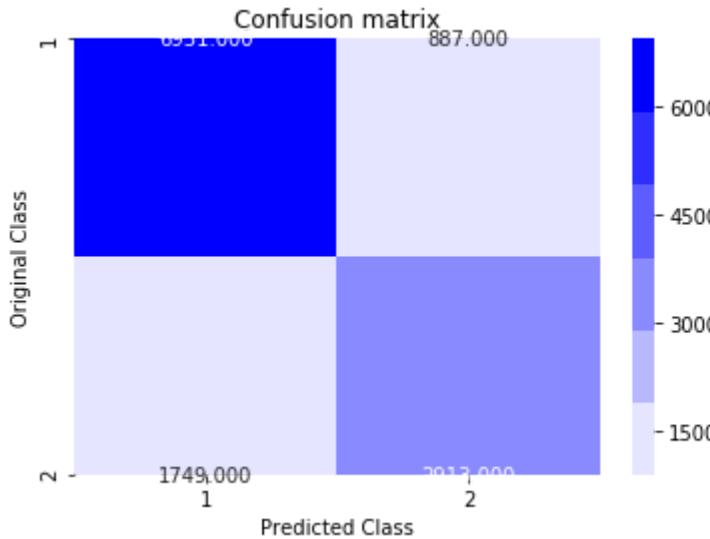
bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, verbose_eval=10)

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))
```



```
[0]      train-logloss:0.685301  valid-logloss:0.685242
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 vectors.
2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

## ▼ 5.1 Reading data from file

```
[500]      train-logloss:0.40145 /  valid-logloss:0.40318
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')
    dfppro = dfppro.drop(['id'],axis=1)

df1.head(2)
```



	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	<b>question2</b>	<b>is_duplicate</b>	<b>cwc_min</b>	<b>cwc_max</b>	<b>csc_min</b>	<b>csc</b>
<b>0</b>	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
<b>1</b>	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	0.599981

2 rows × 21 columns

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

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



	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	<b>question2</b>	<b>is_duplicate</b>	<b>cwc_min</b>	<b>cwc_max</b>	<b>csc_min</b>	<b>csc</b>
<b>0</b>	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
<b>1</b>	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	0.599981

2 rows × 32 columns

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



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

```
[0 rows x 32 columns]
```

```
# Filling the null values with ''
dfnlp = dfnlp.fillna('')
```

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

Empty DataFrame  
Columns: [id, qid1, qid2, question1, question2, is\_duplicate, cwc\_min, cwc\_max, csc\_min, Index: []  
[0 rows x 32 columns]

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

```
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
```

Empty DataFrame  
Columns: [id, qid1, qid2, question1, question2, cwc\_min, cwc\_max, csc\_min, Index: []  
(37500, 31)

```
y_train.shape
```

Empty DataFrame  
Columns: [id, qid1, qid2, question1, question2, cwc\_min, cwc\_max, csc\_min, Index: []  
(37500,)

```
y_test.shape
```

Empty DataFrame  
Columns: [id, qid1, qid2, question1, question2, cwc\_min, cwc\_max, csc\_min, Index: []  
(12500,)

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

Empty DataFrame  
Columns: [id, qid1, qid2, question1, question2, cwc\_min, cwc\_max, csc\_min, Index: []  
(12500, 31)

```
X_train.head()
```

Empty DataFrame  
Columns: [id, qid1, qid2, question1, question2, cwc\_min, cwc\_max, csc\_min, Index: []  
[0 rows x 32 columns]

	<b>id</b>	<b>qid1</b>	<b>qid2</b>	<b>question1</b>	<b>question2</b>	<b>cwc_min</b>	<b>cwc_max</b>	<b>csc_min</b>	<b>csc_max</b>
<b>23561</b>	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
<b>3536</b>	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
<b>33192</b>	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
<b>35725</b>	35725	65244	65245	what is china doing to help nepal	how can we help nepal	0.999950	0.666644	0.000000	0.000000
<b>6320</b>	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

5 rows × 31 columns

## ▼ 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'])
#questions = list(df['question1']) + list(df['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,
                  min_df=10, ngram_range=(1, 2), norm='l2', preprocessor=None,
                  smooth_idf=True, stop_words=None, strip_accents=None,
                  sublinear_tf=False, token_pattern='(\\u)\\b\\w\\w+\\b',
                  tokenizer=None, use_idf=True, vocabulary=None)

```

## ▼ Train Data

```

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

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

```

**👤** Shape of matrix after one hot encoding (37500, 13369)  
 the number of unique words 13369

```

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

```

**👤** Shape of matrix after one hot encoding (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=True)

X_train_feature_df.head(2)

```

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word
<b>23561</b>	0.333322	0.333322	0.749981	0.428565	0.57142	0.399996		0.0
<b>3536</b>	0.000000	0.000000	0.333322	0.166664	0.11111	0.083333		0.0

2 rows × 26 columns

```

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

#so we need to convert our feature data into sparse matrix so that we will combine our feature

```

```
so we need to convert our feature data into sparse matrix so that we will combine our feature
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
```

```
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
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
```

```
----- 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 train 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 train data -----

```
Class 0: 0.37296 Class 1: 0.37296
```

## ▼ 5.4 Function For Confusion Matrix

```

# 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 cl

    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 dia
    # 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 dia
    # 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()
```

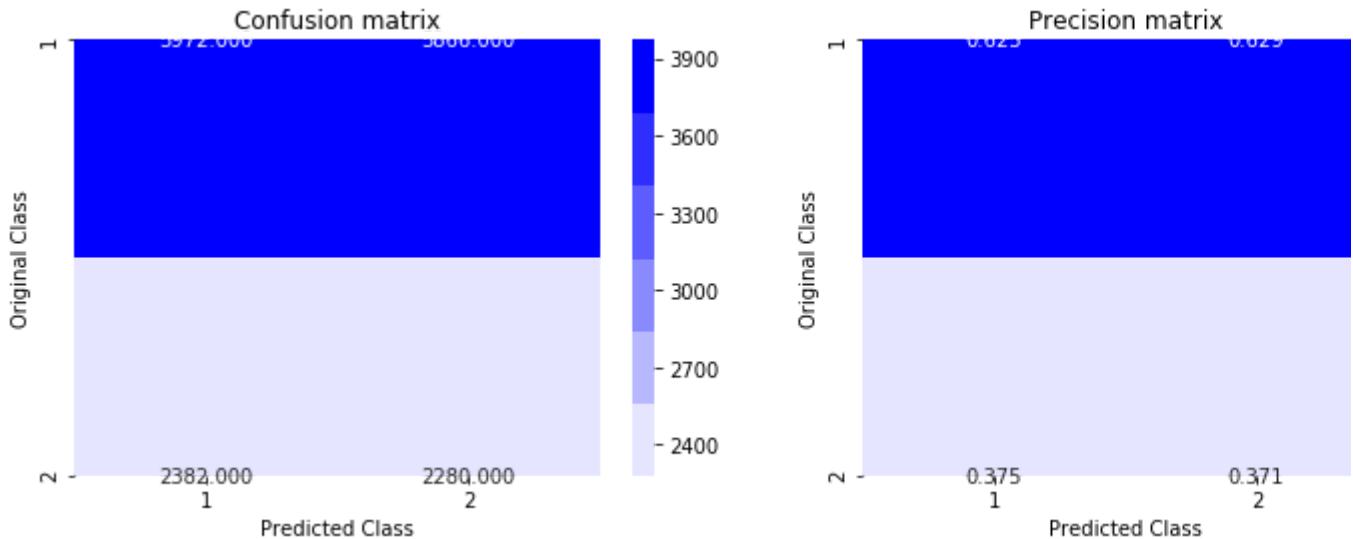
## ▼ 5.5 Building a random model (Finding worst-case log-loss)

```
# 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 0.8878177387261336



## ▼ 5.6 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
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# 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(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_))

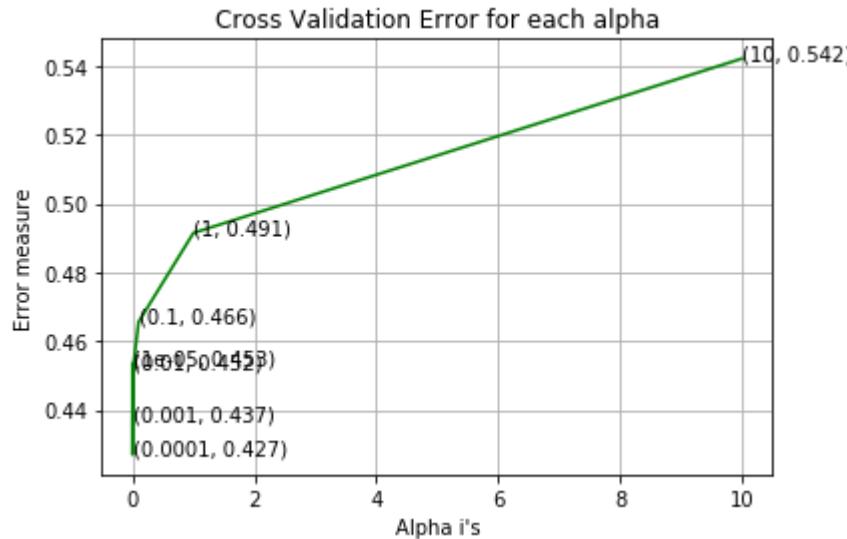
fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='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)
```



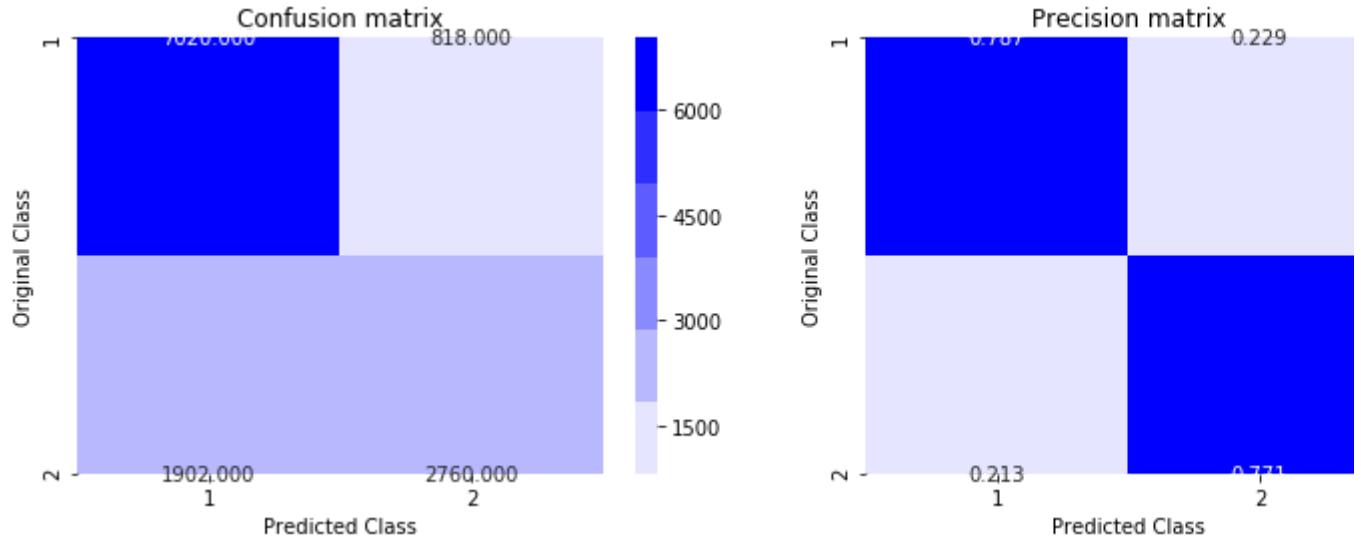
```
For values of alpha = 1e-05 The log loss is: 0.4534364133470595
For values of alpha = 0.0001 The log loss is: 0.427024623201395
For values of alpha = 0.001 The log loss is: 0.43679236027990154
For values of alpha = 0.01 The log loss is: 0.45224537554092165
For values of alpha = 0.1 The log loss is: 0.46560955951780664
For values of alpha = 1 The log loss is: 0.4914680785258957
For values of alpha = 10 The log loss is: 0.5421455955505056
```



For values of best alpha = 0.0001 The train log loss is: 0.41523107644180174

For values of best alpha = 0.0001 The test log loss is: 0.427024623201395

Total number of data points : 12500



## 5.7 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
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
#                 max_iter=1000, tol=0.001, random_state=None, learning_rate='optimal',
#                 eta0=0.0, power_t=0.5, class_weight=None, verbose=0, epsilon=0.1)
```

```
# n_jobs=1, verbose=0, epsilon=0.1, n_iter=1, random_state=None, learning_rate='optimal',
# 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(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_))

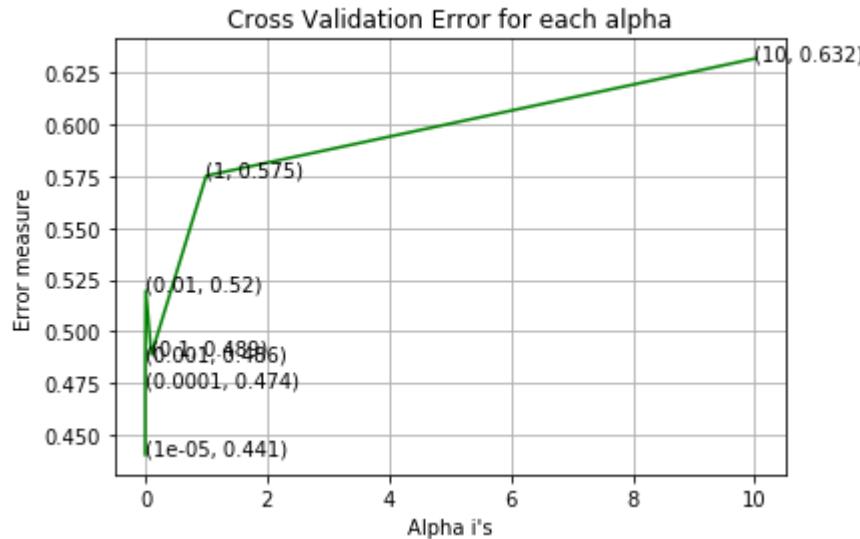
fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

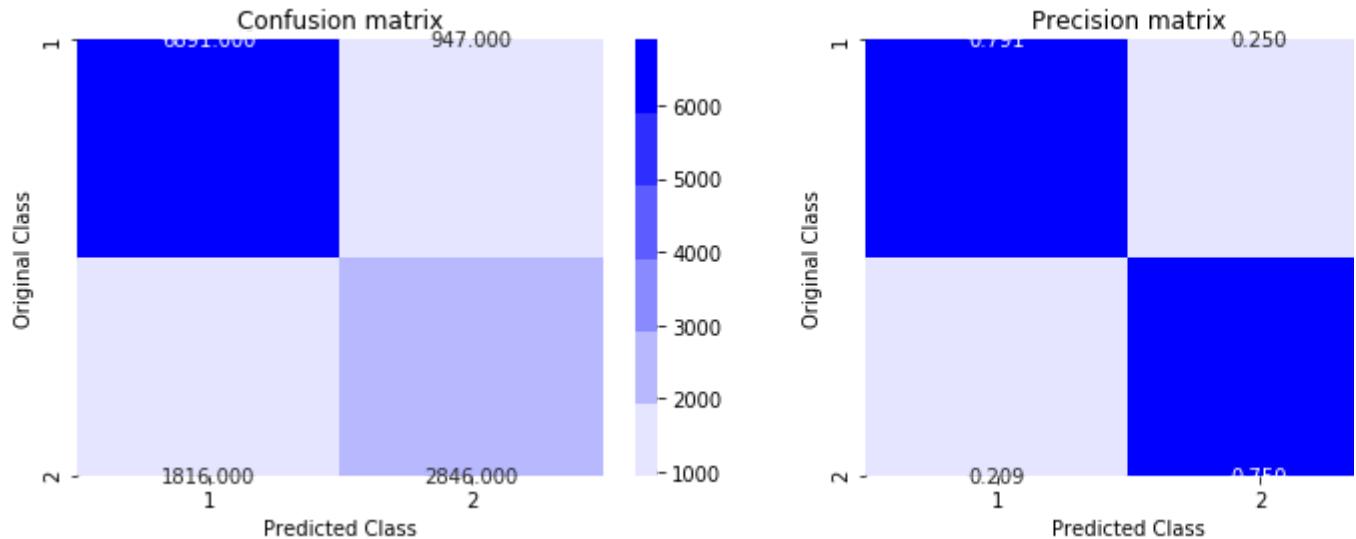
predict_y = sig_clf.predict_proba(tfidf_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_t
predict_y = sig_clf.predict_proba(tfidf_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_te
predicted_y =np.argmax(predict_y, 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
```



## 5.8 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(X_train, y_train)

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


```

## ▼ B. With Best Params

```

bst = xgb.XGBClassifier(max_depth=10,learning_rate=0.1042,objective='binary:logistic',gamma=0
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))

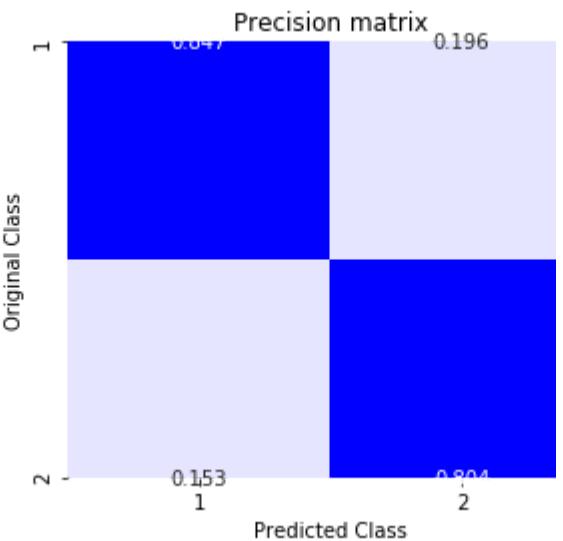
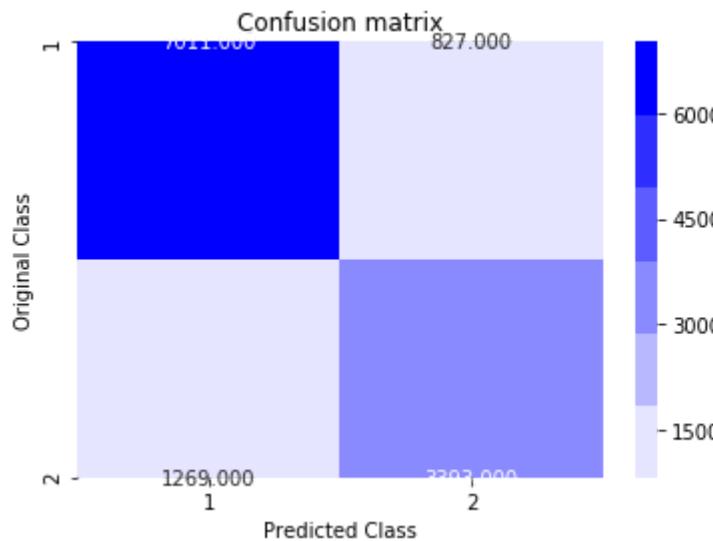
predict_y = clf_calib.predict_proba(tfidf_test)
print("The test log loss is : ",log_loss(y_test, predict_y,labels=bst.classes_, eps=1e-15))

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

```



The train log loss is: 0.2499683753545042  
The test log loss is : 0.3461761557838114



## ▼ TFIDF Weighted Word2Vec

```
# Load Basic Features
dftw_50k = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
#Taking samples of 50k
# Creating duplicate of df_50k for TFIDF Weighted Word2Vec
dftw_50k = dftw_50k.sample(n = 50000)
print("Columns in dftw_50k dataframne:\n")
print(dftw_50k.columns)

dftw_50k.head()
```



Columns in dftw\_50k dataframe:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

|               | <b>id</b> | <b>qid1</b> | <b>qid2</b> | <b>question1</b>                                  | <b>question2</b>                                  | <b>is_duplicate</b> | <b>freq_qid1</b> | <b>freq_q</b> |
|---------------|-----------|-------------|-------------|---------------------------------------------------|---------------------------------------------------|---------------------|------------------|---------------|
| <b>178300</b> | 178300    | 221393      | 273887      | Is Quora degrading itself?                        | Why is the quality of Quora degrading?            | 0                   | 1                |               |
| <b>24373</b>  | 24373     | 12916       | 45545       | What is the best time for studying? Why?          | What is the best time of the day to learn or s... | 0                   | 3                |               |
| <b>65759</b>  | 65759     | 114071      | 114072      | As an international student in the United Stat... | Are international students on an F-1 Visa elig... | 0                   | 1                |               |
| <b>11495</b>  | 11495     | 22194       | 18788       | How can one learn to scrap web data using Python? | What are some good resources to learn web scra... | 1                   | 3                |               |
| <b>13576</b>  | 13576     | 26055       | 26056       | Why it is diffulcut to get jobs in upwork.com?    | Why am I not getting any freelance jobs on Upw... | 0                   | 1                |               |

```
dftw_50k['question1'] = dftw_50k['question1'].apply(lambda x: str(x))
dftw_50k['question2'] = dftw_50k['question2'].apply(lambda x: str(x))
```

```
x_tw = dftw_50k.drop(['is_duplicate', 'id'], axis = 1)
y_tw = dftw_50k['is_duplicate']
```

```
#Train Test Split
from sklearn.model_selection import train_test_split
```

```
x_train_tw, x_test_tw, y_train_tw, y_test_tw = train_test_split(x_tw, y_tw, test_size = 0.3,
```

```
print("Shape of x train data:", x_train_tw.shape)
print("Shape of x test data:", x_test_tw.shape)
```

```
print("Shape of y train data:", y_train_tw.shape)
print("Shape of y test data:", y_test_tw.shape)
```

Shape of x train data: (35000, 15)
Shape of x test data: (15000, 15)
Shape of y train data: (35000,)
Shape of y test data: (15000,)

```
# With train data, creating list of questions, dictionary of feature names and idf values
```

```
# Importing library
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# Merge texts
```

```
questions = list(x_train_tw['question1']) + list(x_train_tw['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_))
```

```
# Defining a function 'vec' to create TF-IDF Weighted Word2Vec
```

```
# Importing libraries
```

```
import os
import spacy
from tqdm import tqdm
```

```
def vec(xtw):
```

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

```
vecs = []
```

```
# https://github.com/noamraph/tqdm
```

```
# tqdm is used to print the progress bar
for qu in tqdm(list(xtw)):
```

```
    doc = nlp(qu)
    # 96 is the number of dimensions of vectors
    mean_vec = np.zeros([len(doc), 96])
```

```
    for word in doc:
        # word2vec
        vec = word.vector
        # fetch df score
```

```
        try:
            idf = word2tfidf[str(word)]
```

```
except:  
    idf = 0  
  
    # compute final vec  
    mean_vec += vec * idf  
  
    mean_vec = mean_vec.mean(axis = 0)  
    vecs.append(mean_vec)  
#dftw_100k['q1_feats_m'] = list(vecs1)  
  
return vecs
```

```
# Calling 'vec' function for train question1
```

```
x_train_tw['que1_tw'] = vec(x_train_tw['question1'])
```

👤 100% |  | 35000/

```
# Calling 'vec' function for train question2
```

```
x_train_tw['que2_tw'] = vec(x_train_tw['question2'])
```

👤 100% |  | 35000/

```
# Calling 'vec' function for test question1
```

```
x_test_tw['que1_tw'] = vec(x_test_tw['question1'])
```

👤 100% |  | 15000/

```
# Calling 'vec' function for test question2
```

```
x_test_tw['que2_tw'] = vec(x_test_tw['question2'])
```

👤 100% |  | 15000/

```
print("Type of x_train_tw['que1_tw']:", type(x_train_tw['que1_tw']))  
print("Type of x_train_tw['que2_tw']:", type(x_train_tw['que2_tw']), '\n')  
print("Type of x_test_tw['que1_tw']:", type(x_test_tw['que1_tw']))  
print("Type of x_test_tw['que2_tw']:", type(x_test_tw['que2_tw']), '\n')
```

```
print("Shape of x train question1:", x_train_tw['que1_tw'].shape)  
print("Shape of x test question1 data:", x_test_tw['que1_tw'].shape, '\n')
```

```
print("Shape of x train question2:", x_train_tw['que2_tw'].shape)  
print("Shape of x test question2 data:", x_test_tw['que1_tw'].shape, '\n')
```



```
Type of x_train_tw['que1_tw']: <class 'pandas.core.series.Series'>
Type of x_train_tw['que2_tw']: <class 'pandas.core.series.Series'>
```

```
Type of x_test_tw['que1_tw']: <class 'pandas.core.series.Series'>
Type of x_test_tw['que2_tw']: <class 'pandas.core.series.Series'>
```

```
Shape of x train question1: (35000,)
```

```
Shape of x test question1 data: (15000,)
```

```
Shape of x train question2: (35000,)
```

```
Shape of x test question2 data: (15000,)
```

```
# Train dataframe
```

```
x_tr_tw1 = pd.DataFrame(x_train_tw['que1_tw'].values.tolist(), index = x_train_tw.index)
```

```
x_tr_tw2 = pd.DataFrame(x_train_tw['que2_tw'].values.tolist(), index = x_train_tw.index,
columns = np.arange(x_tr_tw1.shape[1], x_tr_tw1.shape[1] * 2))
```

```
# Test dataframe
```

```
x_te_tw1 = pd.DataFrame(x_test_tw['que1_tw'].values.tolist(), index = x_test_tw.index)
```

```
x_te_tw2 = pd.DataFrame(x_test_tw['que2_tw'].values.tolist(), index = x_test_tw.index,
columns = np.arange(x_te_tw1.shape[1], x_te_tw1.shape[1] * 2))
```

```
#Concatinating train question1 and train question2 vectors with dataframe
```

```
final_tr_tw = pd.concat([x_train_tw, x_tr_tw1, x_tr_tw2], axis = 1)
```

```
# Dropping question1 and question2 columns from final_test dataframe
```

```
final_te_tw = pd.concat([x_test_tw, x_te_tw1, x_te_tw2], axis = 1)
```

```
# Filling train dataframe
```

```
final_tr_tw = final_tr_tw.fillna(0)
```

```
# Filling test dataframe
```

```
final_te_tw = final_te_tw.fillna(0)
```

```
# Dropping question1 and question2 columns from final_train dataframe
```

```
final_tr_tw = final_tr_tw.drop(['question1', 'question2', 'que1_tw', 'que2_tw'], axis = 1)
```

```
# Dropping question1 and question2 columns from final_test dataframe
```

```
final_te_tw = final_te_tw.drop(['question1', 'question2', 'que1_tw', 'que2_tw'], axis = 1)
```

```
print("Shape of final_tr_tw dataframe:", final_tr_tw.shape, '\n')
```

```
print("Shape of final_te_tw dataframe:", final_te_tw.shape, '\n')
```

Shape of final\_tr\_tw dataframe: (35000, 205)

Shape of final\_te\_tw dataframe: (15000, 205)

```
# Saving final train data
```

```
final_tr_tw.to_csv("quora_final_tr_tw.csv")
```

```
# Saving final test data
```

```
final_te_tw.to_csv("quora_final_te_tw.csv")
```

```
# Import libraries
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
from xgboost import XGBClassifier
```

```
from sklearn.metrics import log_loss
```

```
start = dt.now()
```

```
# Parameters we need to try are
```

```
param_grid = {'n_estimators' : [5, 10, 100, 500], 'max_depth' : [2, 5, 8, 10]}
```

```
rs_k = RandomizedSearchCV(estimator = XGBClassifier(objective = 'binary:logistic', eval_metric = 'logloss'), param_distributions = param_grid)
```

```
# fit train sets
```

```
rs_k.fit(final_tr_tw, y_train_tw)
```

```
# Prediction
```

```
predict_tw = rs_k.predict(final_te_tw)
```

```
print("Time taken to run this cell:", dt.now() - start)
```

Time taken to run this cell: 0:22:35.269475

```
bp = rs_k.best_params_
```

```
bs = rs_k.best_score_
```

```
print("Optimal hyperParameter:", bp, '\n')
```

```
print("Maximum accuracy:", bs * 100)
```

Optimal hyperParameter: {'n\_estimators': 500, 'max\_depth': 5}

Maximum accuracy: 79.9057142857143

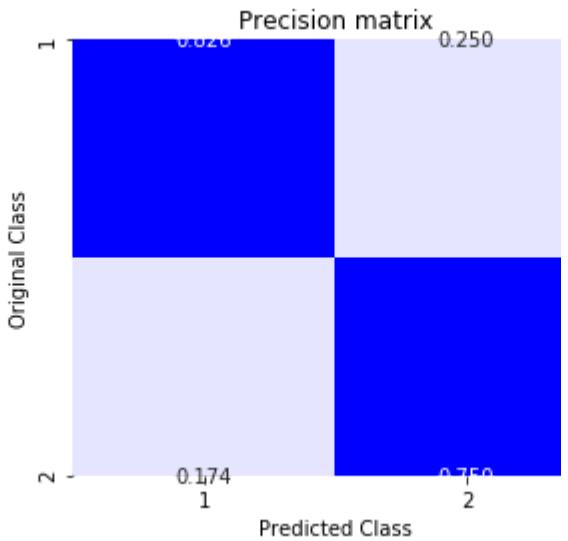
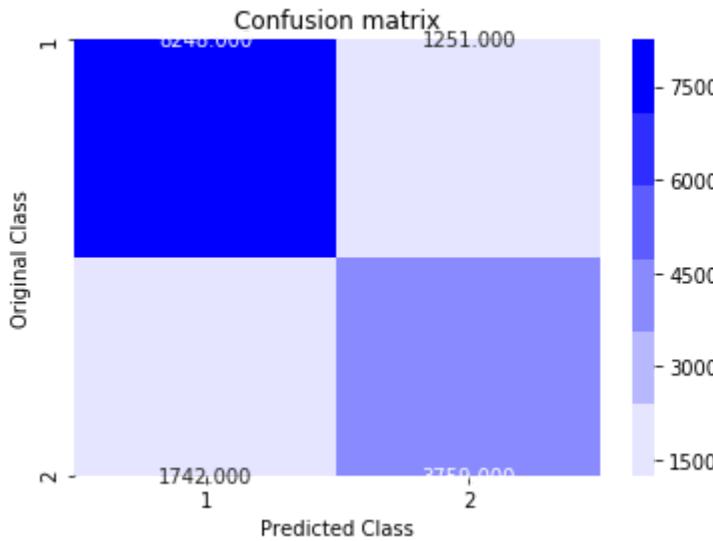
## ▼ Confusion Matrix

```
predicted_y = np.array(predict_tw > 0.5, dtype = int)

print("Total number of data points :", len(predicted_y))

plot_confusion_matrix(y_test_tw, predicted_y)
```

👤 Total number of data points : 15000



## ▼ Hyperparameters

max\_depth: 10

n\_estimators: 100

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 10
params['n_estimators'] = 100

d_train = xgb.DMatrix(final_tr_tw, label= y_train_tw)
d_test = xgb.DMatrix(final_te_tw, label = y_test_tw)

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

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

xgdm = xgb.DMatrix(final_tr_tw,y_train_tw)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test_tw, predict_y, eps=1e-15))
```



```
[0]      train-logloss:0.682685  valid-logloss:0.684053
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.
[1]      train-logloss:0.67253   valid-logloss:0.675285
[2]      train-logloss:0.662892  valid-logloss:0.666873
[3]      train-logloss:0.653559  valid-logloss:0.658784
[4]      train-logloss:0.644464  valid-logloss:0.651093
[5]      train-logloss:0.635684  valid-logloss:0.643558
[6]      train-logloss:0.627259  valid-logloss:0.636413
[7]      train-logloss:0.619013  valid-logloss:0.629573
[8]      train-logloss:0.611035  valid-logloss:0.622898
[9]      train-logloss:0.603278  valid-logloss:0.616472
[10]     train-logloss:0.595753  valid-logloss:0.61029
[11]     train-logloss:0.588505  valid-logloss:0.604363
[12]     train-logloss:0.581432  valid-logloss:0.598615
[13]     train-logloss:0.574594  valid-logloss:0.593007
[14]     train-logloss:0.567993  valid-logloss:0.587659
[15]     train-logloss:0.561445  valid-logloss:0.582469
[16]     train-logloss:0.555151  valid-logloss:0.577451
[17]     train-logloss:0.549024  valid-logloss:0.572565
[18]     train-logloss:0.543019  valid-logloss:0.567842
[19]     train-logloss:0.537064  valid-logloss:0.56314
[20]     train-logloss:0.531377  valid-logloss:0.55863
[21]     train-logloss:0.525739  valid-logloss:0.554304
[22]     train-logloss:0.52028   valid-logloss:0.550101
[23]     train-logloss:0.514974  valid-logloss:0.545981
[24]     train-logloss:0.509765  valid-logloss:0.542011
[25]     train-logloss:0.50476   valid-logloss:0.538137
[26]     train-logloss:0.499839  valid-logloss:0.534365
[27]     train-logloss:0.495086  valid-logloss:0.530792
[28]     train-logloss:0.490457  valid-logloss:0.527273
[29]     train-logloss:0.485944  valid-logloss:0.523819
[30]     train-logloss:0.481496  valid-logloss:0.520523
[31]     train-logloss:0.477217  valid-logloss:0.517267
[32]     train-logloss:0.473     valid-logloss:0.51418
[33]     train-logloss:0.468961  valid-logloss:0.51119
[34]     train-logloss:0.464915  valid-logloss:0.508262
[35]     train-logloss:0.460966  valid-logloss:0.505378
[36]     train-logloss:0.457182  valid-logloss:0.502618
[37]     train-logloss:0.453382  valid-logloss:0.499851
[38]     train-logloss:0.449714  valid-logloss:0.497267
[39]     train-logloss:0.44608   valid-logloss:0.494721
[40]     train-logloss:0.442572  valid-logloss:0.492224
[41]     train-logloss:0.439085  valid-logloss:0.489819
[42]     train-logloss:0.43565   valid-logloss:0.487432
[43]     train-logloss:0.432245  valid-logloss:0.485088
[44]     train-logloss:0.428874  valid-logloss:0.482701
[45]     train-logloss:0.425653  valid-logloss:0.480604
[46]     train-logloss:0.422496  valid-logloss:0.478283
[47]     train-logloss:0.419389  valid-logloss:0.476099
[48]     train-logloss:0.416354  valid-logloss:0.474079
[49]     train-logloss:0.413382  valid-logloss:0.472049
[50]     train-logloss:0.410398  valid-logloss:0.470139
[51]     train-logloss:0.407533  valid-logloss:0.468284
[52]     train-logloss:0.404719  valid-logloss:0.466486
[53]     train-logloss:0.401996  valid-logloss:0.46472
```

```
[54]    train-logloss:0.399303  valid-logloss:0.462962
[55]    train-logloss:0.396646  valid-logloss:0.461283
[56]    train-logloss:0.394059  valid-logloss:0.45962
[57]    train-logloss:0.391522  valid-logloss:0.45803
[58]    train-logloss:0.389056  valid-logloss:0.45644
[59]    train-logloss:0.386633  valid-logloss:0.454882
[60]    train-logloss:0.384297  valid-logloss:0.453363
[61]    train-logloss:0.382015  valid-logloss:0.451887
[62]    train-logloss:0.379661  valid-logloss:0.450474
[63]    train-logloss:0.377392  valid-logloss:0.449074
[64]    train-logloss:0.375168  valid-logloss:0.447711
[65]    train-logloss:0.372913  valid-logloss:0.446429
[66]    train-logloss:0.370701  valid-logloss:0.445206
[67]    train-logloss:0.368554  valid-logloss:0.444023
[68]    train-logloss:0.366584  valid-logloss:0.442838
[69]    train-logloss:0.36458  valid-logloss:0.441667
[70]    train-logloss:0.362554  valid-logloss:0.440584
[71]    train-logloss:0.360748  valid-logloss:0.439392
[72]    train-logloss:0.358862  valid-logloss:0.438399
[73]    train-logloss:0.356994  valid-logloss:0.437302
[74]    train-logloss:0.355162  valid-logloss:0.436234
[75]    train-logloss:0.353305  valid-logloss:0.435263
[76]    train-logloss:0.351425  valid-logloss:0.434247
[77]    train-logloss:0.349686  valid-logloss:0.433214
[78]    train-logloss:0.348096  valid-logloss:0.432245
[79]    train-logloss:0.346424  valid-logloss:0.431233
[80]    train-logloss:0.34479  valid-logloss:0.430294
[81]    train-logloss:0.343125  valid-logloss:0.429348
[82]    train-logloss:0.341546  valid-logloss:0.428472
[83]    train-logloss:0.339961  valid-logloss:0.427578
[84]    train-logloss:0.338388  valid-logloss:0.42669
[85]    train-logloss:0.336835  valid-logloss:0.425833
[86]    train-logloss:0.335318  valid-logloss:0.425011
[87]    train-logloss:0.333825  valid-logloss:0.424196
[88]    train-logloss:0.332336  valid-logloss:0.423408
[89]    train-logloss:0.330863  valid-logloss:0.422686
[90]    train-logloss:0.329486  valid-logloss:0.421926
[91]    train-logloss:0.328064  valid-logloss:0.421188
[92]    train-logloss:0.326771  valid-logloss:0.420464
[93]    train-logloss:0.325344  valid-logloss:0.419783
[94]    train-logloss:0.324076  valid-logloss:0.419093
[95]    train-logloss:0.322823  valid-logloss:0.418508
[96]    train-logloss:0.321539  valid-logloss:0.4179
[97]    train-logloss:0.320303  valid-logloss:0.417309
[98]    train-logloss:0.319071  valid-logloss:0.41672
[99]    train-logloss:0.317833  valid-logloss:0.416177
[100]   train-logloss:0.316595  valid-logloss:0.415587
[101]   train-logloss:0.315349  valid-logloss:0.414976
[102]   train-logloss:0.314238  valid-logloss:0.414469
[103]   train-logloss:0.313016  valid-logloss:0.413891
[104]   train-logloss:0.31181  valid-logloss:0.41331
[105]   train-logloss:0.310747  valid-logloss:0.412749
[106]   train-logloss:0.309699  valid-logloss:0.412272
[107]   train-logloss:0.308621  valid-logloss:0.411723
[108]   train-logloss:0.307509  valid-logloss:0.411274
[109]   train-logloss:0.30646  valid-logloss:0.410735
[110]   train-logloss:0.305382  valid-logloss:0.410273
[111]   train-logloss:0.304272  valid-logloss:0.409703
```

```
[112] train-logloss:0.30335  valid-logloss:0.409228
[113] train-logloss:0.302329  valid-logloss:0.408854
[114] train-logloss:0.301327  valid-logloss:0.408423
[115] train-logloss:0.300303  valid-logloss:0.407959
[116] train-logloss:0.299282  valid-logloss:0.407586
[117] train-logloss:0.298301  valid-logloss:0.40721
[118] train-logloss:0.297421  valid-logloss:0.406785
[119] train-logloss:0.296551  valid-logloss:0.406378
[120] train-logloss:0.295745  valid-logloss:0.405988
[121] train-logloss:0.294934  valid-logloss:0.405606
[122] train-logloss:0.294109  valid-logloss:0.405217
[123] train-logloss:0.293231  valid-logloss:0.404829
[124] train-logloss:0.292456  valid-logloss:0.404471
[125] train-logloss:0.291625  valid-logloss:0.404104
[126] train-logloss:0.290871  valid-logloss:0.403776
[127] train-logloss:0.29004   valid-logloss:0.403414
[128] train-logloss:0.289087  valid-logloss:0.40303
[129] train-logloss:0.288224  valid-logloss:0.402676
[130] train-logloss:0.287363  valid-logloss:0.402354
[131] train-logloss:0.286479  valid-logloss:0.402083
[132] train-logloss:0.285495  valid-logloss:0.40178
[133] train-logloss:0.284837  valid-logloss:0.401517
[134] train-logloss:0.284067  valid-logloss:0.401212
[135] train-logloss:0.283391  valid-logloss:0.40096
[136] train-logloss:0.282617  valid-logloss:0.400671
[137] train-logloss:0.282022  valid-logloss:0.400446
[138] train-logloss:0.281146  valid-logloss:0.400242
[139] train-logloss:0.280459  valid-logloss:0.399963
[140] train-logloss:0.279844  valid-logloss:0.399714
[141] train-logloss:0.279051  valid-logloss:0.399477
[142] train-logloss:0.278507  valid-logloss:0.399266
[143] train-logloss:0.277839  valid-logloss:0.398971
[144] train-logloss:0.277079  valid-logloss:0.398646
[145] train-logloss:0.276369  valid-logloss:0.39843
[146] train-logloss:0.275612  valid-logloss:0.398246
[147] train-logloss:0.274823  valid-logloss:0.398057
[148] train-logloss:0.274298  valid-logloss:0.397866
[149] train-logloss:0.273709  valid-logloss:0.397681
[150] train-logloss:0.27293   valid-logloss:0.397517
[151] train-logloss:0.272151  valid-logloss:0.397261
[152] train-logloss:0.271426  valid-logloss:0.397066
[153] train-logloss:0.270865  valid-logloss:0.396843
[154] train-logloss:0.270118  valid-logloss:0.396683
[155] train-logloss:0.269527  valid-logloss:0.396535
[156] train-logloss:0.269075  valid-logloss:0.396387
[157] train-logloss:0.268439  valid-logloss:0.396195
[158] train-logloss:0.267721  valid-logloss:0.396002
[159] train-logloss:0.267266  valid-logloss:0.39584
[160] train-logloss:0.266538  valid-logloss:0.395623
[161] train-logloss:0.26613   valid-logloss:0.395518
[162] train-logloss:0.265552  valid-logloss:0.395356
[163] train-logloss:0.264821  valid-logloss:0.39523
[164] train-logloss:0.263946  valid-logloss:0.395119
[165] train-logloss:0.263408  valid-logloss:0.394924
[166] train-logloss:0.262751  valid-logloss:0.394793
[167] train-logloss:0.261787  valid-logloss:0.394592
[168] train-logloss:0.261166  valid-logloss:0.394473
[169] train-logloss:0.260361  valid-logloss:0.394218
```

```
[105]    train-logloss:0.250504    valid-logloss:0.394510
[170]    train-logloss:0.259771    valid-logloss:0.394232
[171]    train-logloss:0.25894    valid-logloss:0.394044
[172]    train-logloss:0.258359    valid-logloss:0.39388
[173]    train-logloss:0.257789    valid-logloss:0.393816
[174]    train-logloss:0.257203    valid-logloss:0.393706
[175]    train-logloss:0.256653    valid-logloss:0.393558
[176]    train-logloss:0.255742    valid-logloss:0.393369
[177]    train-logloss:0.255265    valid-logloss:0.393276
[178]    train-logloss:0.2547    valid-logloss:0.393142
[179]    train-logloss:0.25424    valid-logloss:0.392989
[180]    train-logloss:0.253489    valid-logloss:0.392862
[181]    train-logloss:0.253043    valid-logloss:0.392781
[182]    train-logloss:0.25226    valid-logloss:0.392578
[183]    train-logloss:0.251731    valid-logloss:0.392486
[184]    train-logloss:0.251193    valid-logloss:0.392401
[185]    train-logloss:0.250919    valid-logloss:0.392292
[186]    train-logloss:0.250406    valid-logloss:0.392209
[187]    train-logloss:0.24965    valid-logloss:0.392025
[188]    train-logloss:0.249096    valid-logloss:0.391948
[189]    train-logloss:0.248622    valid-logloss:0.391836
[190]    train-logloss:0.248228    valid-logloss:0.391776
[191]    train-logloss:0.247433    valid-logloss:0.391612
[192]    train-logloss:0.247065    valid-logloss:0.391523
[193]    train-logloss:0.246409    valid-logloss:0.391372
[194]    train-logloss:0.245999    valid-logloss:0.391286
[195]    train-logloss:0.245306    valid-logloss:0.391145
[196]    train-logloss:0.244666    valid-logloss:0.391036
[197]    train-logloss:0.24425    valid-logloss:0.390999
[198]    train-logloss:0.243512    valid-logloss:0.390908
[199]    train-logloss:0.243093    valid-logloss:0.390873
[200]    train-logloss:0.242677    valid-logloss:0.390768
[201]    train-logloss:0.242291    valid-logloss:0.390708
[202]    train-logloss:0.241887    valid-logloss:0.390643
[203]    train-logloss:0.24118    valid-logloss:0.390538
[204]    train-logloss:0.240722    valid-logloss:0.390456
[205]    train-logloss:0.239998    valid-logloss:0.390349
[206]    train-logloss:0.239611    valid-logloss:0.390276
[207]    train-logloss:0.239126    valid-logloss:0.390215
[208]    train-logloss:0.238727    valid-logloss:0.39017
[209]    train-logloss:0.237982    valid-logloss:0.390045
[210]    train-logloss:0.237634    valid-logloss:0.389966
[211]    train-logloss:0.236992    valid-logloss:0.389863
[212]    train-logloss:0.236288    valid-logloss:0.3898
[213]    train-logloss:0.235841    valid-logloss:0.389745
[214]    train-logloss:0.235549    valid-logloss:0.389702
[215]    train-logloss:0.234823    valid-logloss:0.389598
[216]    train-logloss:0.234442    valid-logloss:0.389566
[217]    train-logloss:0.234119    valid-logloss:0.389555
[218]    train-logloss:0.233409    valid-logloss:0.389488
[219]    train-logloss:0.233144    valid-logloss:0.389484
[220]    train-logloss:0.232622    valid-logloss:0.389442
[221]    train-logloss:0.23212    valid-logloss:0.389391
[222]    train-logloss:0.231483    valid-logloss:0.389297
[223]    train-logloss:0.231096    valid-logloss:0.389242
[224]    train-logloss:0.230586    valid-logloss:0.38922
[225]    train-logloss:0.22995    valid-logloss:0.389138
[226]    train-logloss:0.229366    valid-logloss:0.389024
```

```
[227] train-logloss:0.228741  valid-logloss:0.388969
[228] train-logloss:0.228398  valid-logloss:0.38896
[229] train-logloss:0.227902  valid-logloss:0.388923
[230] train-logloss:0.227445  valid-logloss:0.388833
[231] train-logloss:0.22695   valid-logloss:0.388815
[232] train-logloss:0.226408  valid-logloss:0.388734
[233] train-logloss:0.225778  valid-logloss:0.388676
[234] train-logloss:0.225258  valid-logloss:0.388663
[235] train-logloss:0.224727  valid-logloss:0.388646
[236] train-logloss:0.224374  valid-logloss:0.388638
[237] train-logloss:0.223593  valid-logloss:0.388575
[238] train-logloss:0.223021  valid-logloss:0.38854
[239] train-logloss:0.222505  valid-logloss:0.388518
[240] train-logloss:0.222035  valid-logloss:0.388484
[241] train-logloss:0.221485  valid-logloss:0.388453
[242] train-logloss:0.221255  valid-logloss:0.388428
[243] train-logloss:0.220813  valid-logloss:0.388405
[244] train-logloss:0.22055   valid-logloss:0.388366
[245] train-logloss:0.220265  valid-logloss:0.388314
[246] train-logloss:0.220081  valid-logloss:0.388312
[247] train-logloss:0.21961   valid-logloss:0.388281
[248] train-logloss:0.21924   valid-logloss:0.38826
[249] train-logloss:0.218881  valid-logloss:0.388246
[250] train-logloss:0.218606  valid-logloss:0.388202
[251] train-logloss:0.218284  valid-logloss:0.388155
[252] train-logloss:0.217865  valid-logloss:0.388161
[253] train-logloss:0.217197  valid-logloss:0.388075
[254] train-logloss:0.216612  valid-logloss:0.388042
[255] train-logloss:0.216463  valid-logloss:0.38804
[256] train-logloss:0.215803  valid-logloss:0.387966
[257] train-logloss:0.215263  valid-logloss:0.387881
[258] train-logloss:0.214994  valid-logloss:0.387868
[259] train-logloss:0.214419  valid-logloss:0.387839
[260] train-logloss:0.214045  valid-logloss:0.387815
[261] train-logloss:0.213791  valid-logloss:0.387806
[262] train-logloss:0.21316   valid-logloss:0.387725
[263] train-logloss:0.212793  valid-logloss:0.387721
[264] train-logloss:0.212604  valid-logloss:0.387699
[265] train-logloss:0.211988  valid-logloss:0.387662
[266] train-logloss:0.211469  valid-logloss:0.387623
[267] train-logloss:0.211279  valid-logloss:0.387617
[268] train-logloss:0.210935  valid-logloss:0.38757
[269] train-logloss:0.210313  valid-logloss:0.3875
[270] train-logloss:0.209839  valid-logloss:0.387475
[271] train-logloss:0.209561  valid-logloss:0.387474
[272] train-logloss:0.209172  valid-logloss:0.38743
[273] train-logloss:0.208657  valid-logloss:0.387419
[274] train-logloss:0.208407  valid-logloss:0.387401
[275] train-logloss:0.20804   valid-logloss:0.387429
[276] train-logloss:0.207741  valid-logloss:0.387376
[277] train-logloss:0.207496  valid-logloss:0.387351
[278] train-logloss:0.20717   valid-logloss:0.387348
[279] train-logloss:0.206927  valid-logloss:0.38731
[280] train-logloss:0.2067   valid-logloss:0.387268
[281] train-logloss:0.206364  valid-logloss:0.387248
[282] train-logloss:0.205642  valid-logloss:0.387232
[283] train-logloss:0.20529   valid-logloss:0.387201
[284] train-logloss:0.204483  valid-logloss:0.387178
```

```
[285]  train-logloss:0.204336  valid-logloss:0.387119
[286]  train-logloss:0.203894  valid-logloss:0.387031
[287]  train-logloss:0.203482  valid-logloss:0.387013
[288]  train-logloss:0.202893  valid-logloss:0.386976
[289]  train-logloss:0.202615  valid-logloss:0.386968
[290]  train-logloss:0.202126  valid-logloss:0.387012
[291]  train-logloss:0.201267  valid-logloss:0.38702
[292]  train-logloss:0.200807  valid-logloss:0.38704
[293]  train-logloss:0.200076  valid-logloss:0.387035
[294]  train-logloss:0.199932  valid-logloss:0.387012
[295]  train-logloss:0.199435  valid-logloss:0.387015
[296]  train-logloss:0.198653  valid-logloss:0.387016
[297]  train-logloss:0.197922  valid-logloss:0.386999
[298]  train-logloss:0.197461  valid-logloss:0.386982
[299]  train-logloss:0.196791  valid-logloss:0.387029
[300]  train-logloss:0.196508  valid-logloss:0.386996
[301]  train-logloss:0.195926  valid-logloss:0.386994
[302]  train-logloss:0.195483  valid-logloss:0.386987
[303]  train-logloss:0.195214  valid-logloss:0.386971
[304]  train-logloss:0.194889  valid-logloss:0.386958
[305]  train-logloss:0.194451  valid-logloss:0.386978
[306]  train-logloss:0.194385  valid-logloss:0.386971
[307]  train-logloss:0.193693  valid-logloss:0.386942
[308]  train-logloss:0.193299  valid-logloss:0.386925
[309]  train-logloss:0.192569  valid-logloss:0.386918
[310]  train-logloss:0.192235  valid-logloss:0.386934
[311]  train-logloss:0.192109  valid-logloss:0.386913
[312]  train-logloss:0.192007  valid-logloss:0.386907
[313]  train-logloss:0.191657  valid-logloss:0.3869
[314]  train-logloss:0.19108  valid-logloss:0.3869
[315]  train-logloss:0.19051  valid-logloss:0.386926
[316]  train-logloss:0.189884  valid-logloss:0.386876
[317]  train-logloss:0.189505  valid-logloss:0.386886
[318]  train-logloss:0.189045  valid-logloss:0.386854
[319]  train-logloss:0.188499  valid-logloss:0.38686
[320]  train-logloss:0.187904  valid-logloss:0.38679
[321]  train-logloss:0.187308  valid-logloss:0.386827
[322]  train-logloss:0.186952  valid-logloss:0.386855
[323]  train-logloss:0.186707  valid-logloss:0.386864
[324]  train-logloss:0.186333  valid-logloss:0.386886
[325]  train-logloss:0.186066  valid-logloss:0.386881
[326]  train-logloss:0.185608  valid-logloss:0.386896
[327]  train-logloss:0.185189  valid-logloss:0.386866
[328]  train-logloss:0.184989  valid-logloss:0.386824
[329]  train-logloss:0.184276  valid-logloss:0.386828
[330]  train-logloss:0.183739  valid-logloss:0.386869
[331]  train-logloss:0.182995  valid-logloss:0.386823
[332]  train-logloss:0.182684  valid-logloss:0.386847
[333]  train-logloss:0.182446  valid-logloss:0.386866
[334]  train-logloss:0.181919  valid-logloss:0.386883
[335]  train-logloss:0.181303  valid-logloss:0.386835
[336]  train-logloss:0.180737  valid-logloss:0.386827
[337]  train-logloss:0.180163  valid-logloss:0.386837
[338]  train-logloss:0.179961  valid-logloss:0.38684
[339]  train-logloss:0.179548  valid-logloss:0.386807
[340]  train-logloss:0.179007  valid-logloss:0.386863
```

Stopping. Best iteration:

```
  12201  train-logloss:0.187001  valid-logloss:0.38670
```

The test log loss is: 0.38686106475459336

## ▼ CONCLUSION:

```
from prettytable import PrettyTable
phtable = PrettyTable()
phtable.title = " Model Comparision "
phtable.field_names = ['Serial No.', 'Model Name', 'Tokenizer', 'Hyperparameter Tunning', 'Test'
phtable.add_row(["1", "Random", "TFIDF Weighted W2V", "-", "0.89"])
phtable.add_row(["2", "Logistic Regression", "TFIDF Weighted W2V", "Done", "0.46"])
phtable.add_row(["3", "Linear SVM", "TFIDF Weighted W2V", "Done", "0.46"])
phtable.add_row(["4", "XGBoost", "TFIDF Weighted W2V", "-", "0.399"])
phtable.add_row(["\n", "\n", "\n", "\n", "\n"])
phtable.add_row(["1", "Random", "TFIDF", "-", "0.89"])
phtable.add_row(["2", "Logistic Regression", "TFIDF", "Done", "0.42"])
phtable.add_row(["3", "Linear SVM", "TFIDF", "Done", "0.439"])
phtable.add_row(["4", "XGBoost", "TFIDF", "Done", "0.386"])
print(phtable)
```



| Serial No. | Model Name          | Tokenizer          | Hyperparameter Tunning | Test |
|------------|---------------------|--------------------|------------------------|------|
| 1          | Random              | TFIDF Weighted W2V | -                      |      |
| 2          | Logistic Regression | TFIDF Weighted W2V | Done                   |      |
| 3          | Linear SVM          | TFIDF Weighted W2V | Done                   |      |
| 4          | XGBoost             | TFIDF Weighted W2V | -                      | 0    |
| <hr/>      |                     |                    |                        |      |
| 1          | Random              | TFIDF              | -                      |      |
| 2          | Logistic Regression | TFIDF              | Done                   |      |
| 3          | Linear SVM          | TFIDF              | Done                   | 0    |
| 4          | XGBoost             | TFIDF              | Done                   | 0    |

## ▼ STEP BY STEP PROCEDURE:

1. As we know we have data set which contains Number of rows 404,290, which contains 5 columns : is\_duplicate from which 'is\_duplicate' is a class lable which specify that the question 1 and question 2 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 merged dataframes and got out final matrix. Now after doing simple EDA on dataset we will try some the datset like Frequency of qid1's ,word\_Common and etc. and using this featured datset we will do : rectify which features are most useful features our of all features i.e(which feature is helpful for classif

3. After doing basic Basic feature extractions we will try some Advanced Feature Extraction using NLP we will do Preprocessing of Text and then we will do Advanced Feature Extraction and try to visualise and word clouds.

4. Then we Splitted out data randomly . We could also have done time based splitting, since the mode 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 vectorise word-vectors so that we will be able to apply models on it and after doing all these we will merge all the features + question1 tfidf w2v + and question 2 tfidf w2v. and Now after doing all of these we will apply

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

7. Lets start: here we are there model i.e Logistic Regression linear svm and XgBoost and a random m 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 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 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. We can also see that there is no Overfitting problem , since, Train log-loss and Test log-loss and very close.

11. After that we have applied Linear SVM with hyperparameter tuning. It gave the log-loss of 0.539,we also see that there is no Overfitting problem , since, Train log-loss and Test log-loss and very close.

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

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

