Quora Question Pairs

1. Business Problem

1.1 Description

__ Problem Statement __

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

2. Machine Learning Probelm

2.1 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

- log-loss : https://www.kaggle.com/wiki/LogarithmicLoss
- Binary Confusion Matrix

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

```
!pip install distance
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")
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,accuracy_score, log_loss,precision_recall_curve, auc, roc_curve,normalized_mutual_info_scor
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold ,GridSearchCV, train_test_split
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB,GaussianNB
import math
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier,LogisticRegression

import nltk
nltk.download('stopwords')

!pip install fuzzywuzzy
from fuzzywuzzy import fuzz

from wordcloud import WordCloud
from os import path
from tqdm import tqdm
import spacy
!python -m spacy download en_core_web_lg
```

```
    Requirement already satisfied: distance in /usr/local/lib/python3.10/dist-packages (0.1.3)
    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Package stopwords is already up-to-date!
    Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.10/dist-packages (0.18.0)
    2023-07-23 03:04:26.815954: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
    Collecting en-core-web-lg==3.5.0
      Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_lg-3.5.0/en_core_web_lg-3.5.0-py3-none-any.wh
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 587.7/587.7 MB 2.9 MB/s eta 0:00:00
    Requirement already satisfied: spacy<3.6.0,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from en-core-web-lg==3.5.0) (3.5.4)
    Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->e
    Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->e
    Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-
    Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-w
    Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-
    Requirement already satisfied: thinc<8.2.0,>=8.1.8 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-w
    Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-
    Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-w
    Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-co
    Requirement already satisfied: typer<0.10.0,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-
    Requirement already satisfied: pathy>=0.10.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg=
    Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-c
    Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-w
    Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg=
    Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-co
    Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,
    Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0)
    Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-l
    Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.6.0,>=3.5.0->en-co
    Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<1.
    Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spa
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<
    Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,
    Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.6
    Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spac
    Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.3.0->spacy<3.
    Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->spacy<3.6.0,>=3.5.0->en-co
    ✓ Download and installation successful
    You can now load the package via spacy.load('en_core_web_lg')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

3.1 Reading data and basic stats

```
df = pd.read_csv("/content/drive/MyDrive/train.csv",nrows=25000)
```

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

```
Number of data points: 25000
```

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

```
0    15694
1     9306
Name: is_duplicate, dtype: int64
```

```
df.head()
```

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            25000 non-null  int64 
 1   qid1          25000 non-null  int64 
 2   qid2          25000 non-null  int64 
 3   question1     25000 non-null  object
 4   question2     25000 non-null  object
 5   is_duplicate  25000 non-null  int64 
dtypes: int64(4), object(2)
memory usage: 1.1+ MB
```

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

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

3.2.1 Distribution of data points among output classes

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

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

```
<Axes: xlabel='is_duplicate'>
```

16000
14000
12000

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

```
~> Total number of question pairs for training:
   25000
```

```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_duplicate'].mean()*100, 2)))
```

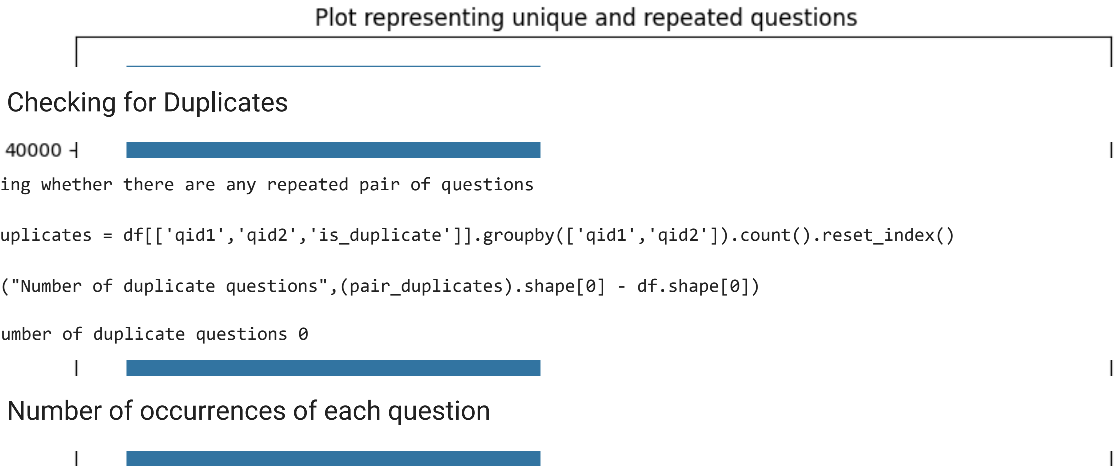
```
~> Question pairs are not Similar (is_duplicate = 0):
   62.78%

~> Question pairs are Similar (is_duplicate = 1):
   37.22%
```

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/unique_qs*

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: 46634

Number of unique questions that appear more than one time: 2606 (5.588197452502467%)

Max number of times a single question is repeated: 10
```

```
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(data=df,x=x,y=y)
plt.show()
```



3.2.3 Checking for Duplicates

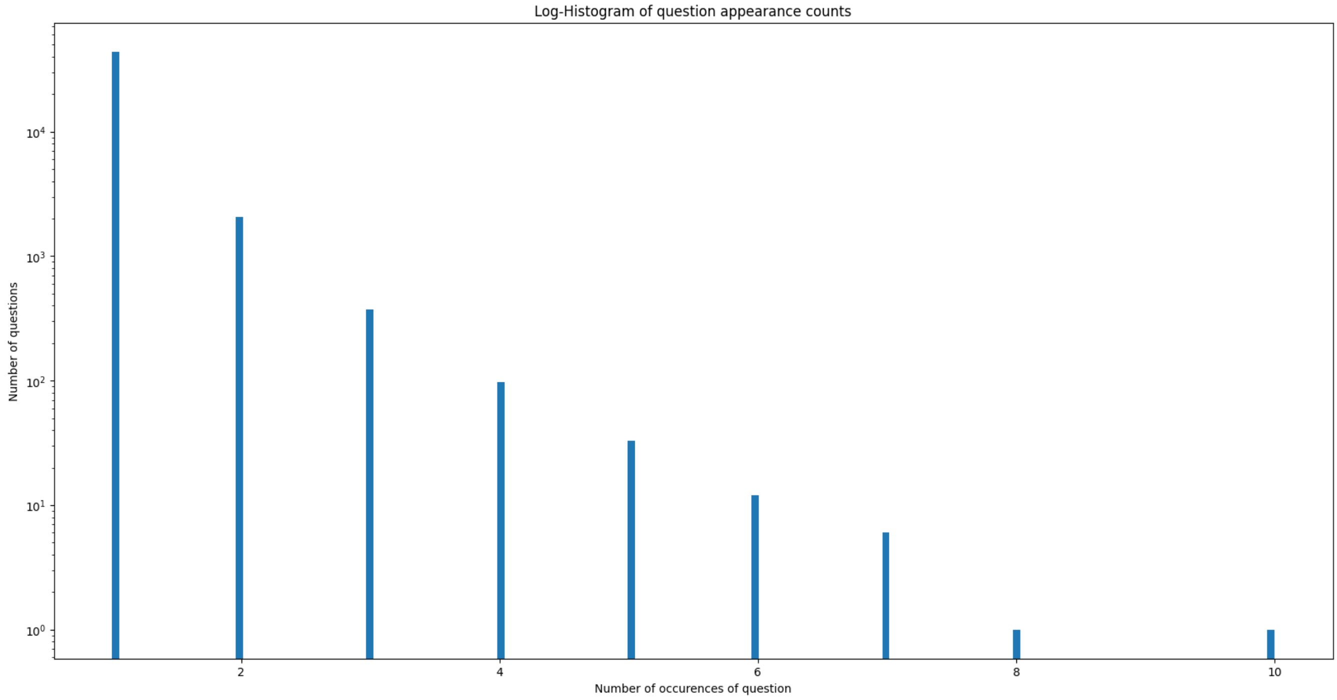
```
#checking whether there are any repeated pair of questions

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

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

```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

```
plt.figure(figsize=(20, 10))

plt.hist(qids.value_counts(), bins=160)

plt.yscale('log', nonpositive='clip')

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

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

plt.ylabel('Number of questions')

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

```
Maximum number of times a single question is repeated: 10
```



3.2.5 Checking for NULL values

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

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

- There are two rows with null values in question2

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

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1
- **q2len** = Length of q2
- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2
- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)
- **word_share** = (word_common)/(word_Total)
- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2
- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

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

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

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

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

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

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

df.head()
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wc
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14	12	10.0	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	1	1	51	88	8	13	4.0	
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	14	10	4.0	
3	3	7	8	Why am I mentally very lonely?	Find the remainder when [math]23^(24)	0	1	1	50	65	11	9	0.0	

```
df.shape
```

```
(25000, 17)
```

3.3.1 Analysis of some of the extracted features

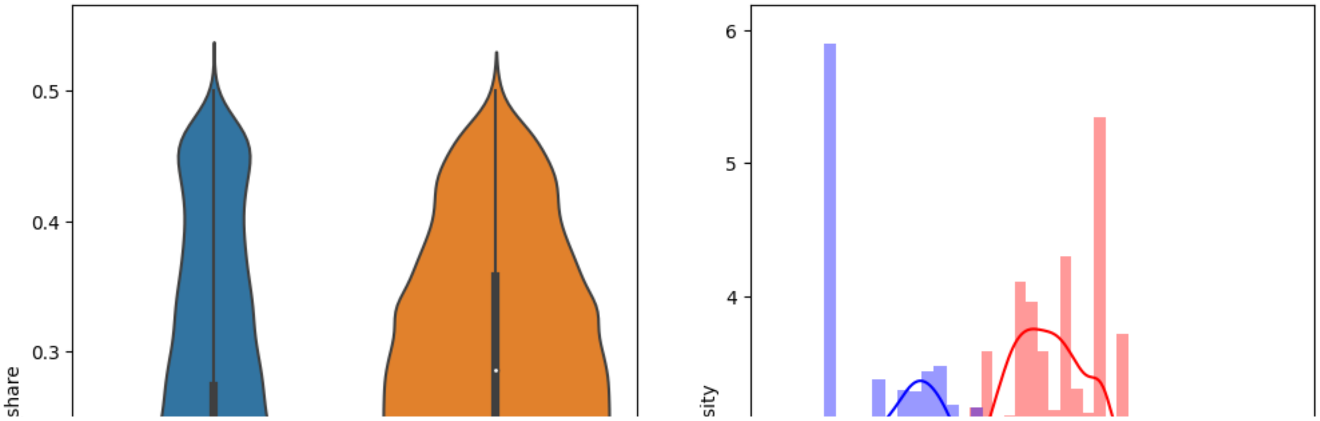
- Here are some questions have only one single words.

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

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

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

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 6
Number of Questions with minimum length [question2] : 1
```

3.3.1.1 Feature: word_share

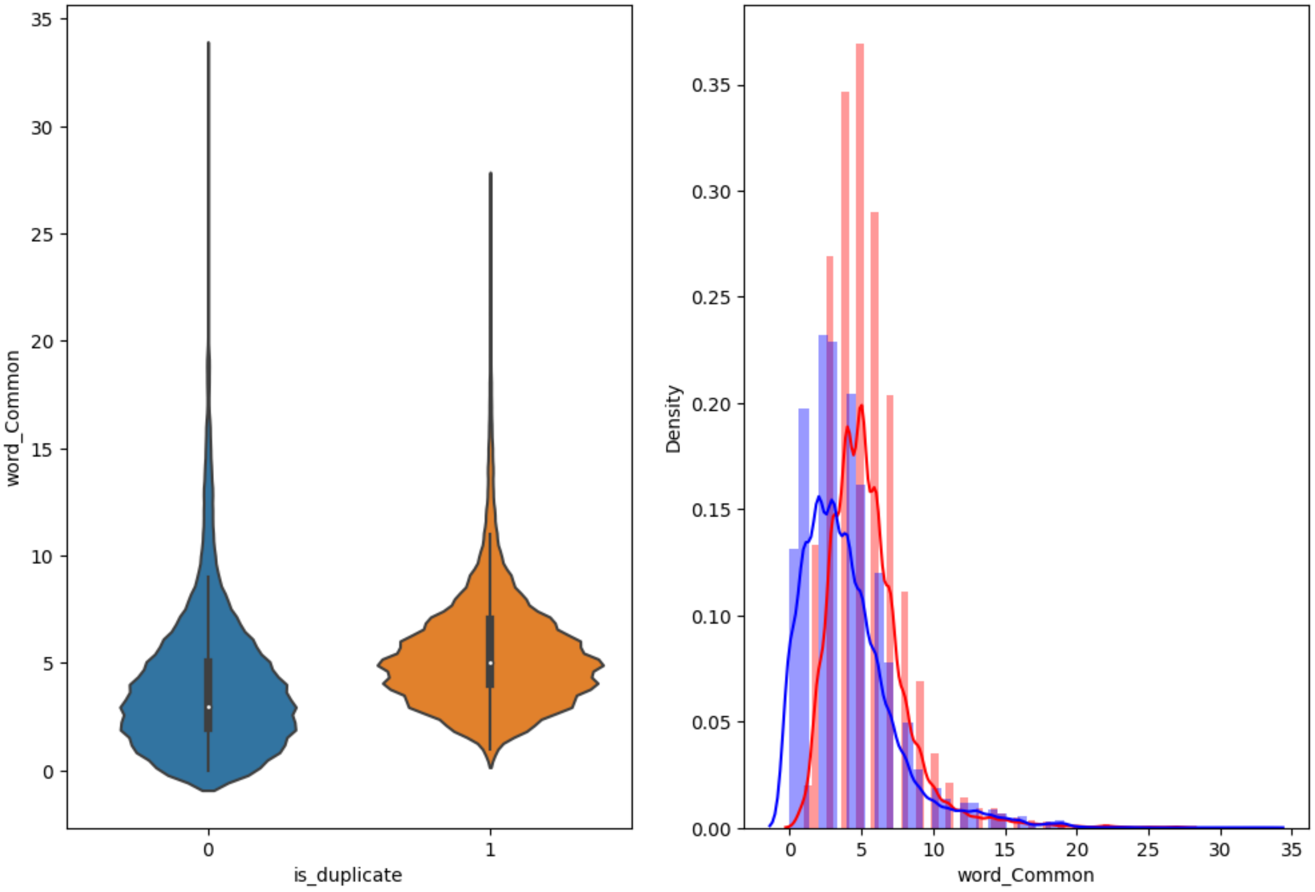
```
plt.figure(figsize=(12, 8))

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

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



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

3.3.1.2 Feature: word_Common

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

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

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



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

1.2.1 : EDA: Advanced Feature Extraction.

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byte-0x9c
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous notebook")
```

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

```
STOP_WORDS = stopwords.words("english")

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


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

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


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


    return x


SAFE_DIV=0.0001
def get_token_features(q1, q2):
    token_features = [0.0]*10

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

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

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

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

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

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


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

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

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

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

# get the Longest Common sub string

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

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

    print("token features...")

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

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


if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("/content/drive/MyDrive/train.csv",nrows=25000)
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

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

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	first_wor
0	0	1	2	what is the step by step guide to invest in	what is the step by step guide to invest in	0	0.999980	0.833319	0.999983	0.999983	...	0.785709	0.0	
1	1	3	4	kohinoor	the indian	0	0.799984	0.399996	0.749981	0.599988		0.466664	0.0	

2 rows × 21 columns

```
print("No of data points and features in data is:",df.shape)
```

```
No of data points and features in data is: (25000, 21)
```

3.5.1.1 Plotting Word clouds

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

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

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

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

```
Number of data points in class 1 (duplicate pairs) : 18612
Number of data points in class 0 (non duplicate pairs) : 31388
```

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

textp_w = open(path.join(d, 'train_p.txt')).read()
textn_w = open(path.join(d, 'train_n.txt')).read()
stopwords = set(STOP_WORDS)
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 : 998399
Total number of words in non duplicate pair questions : 2048708
```

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

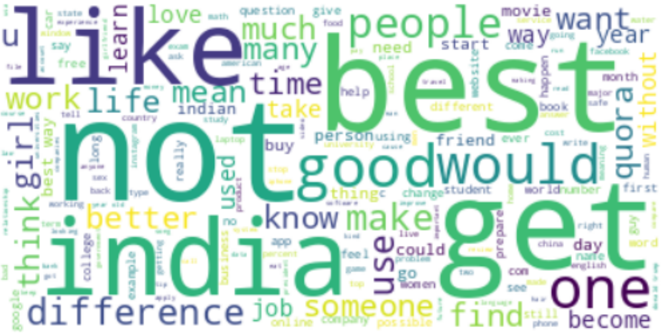
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

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

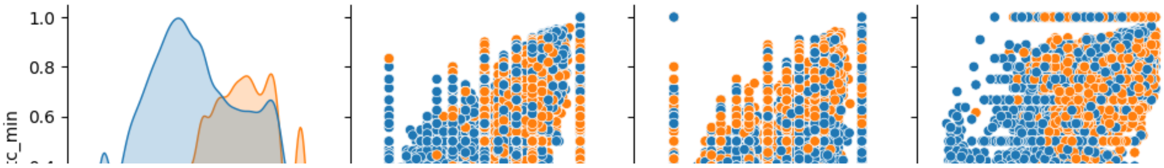
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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



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

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



3.5.2 Visualization

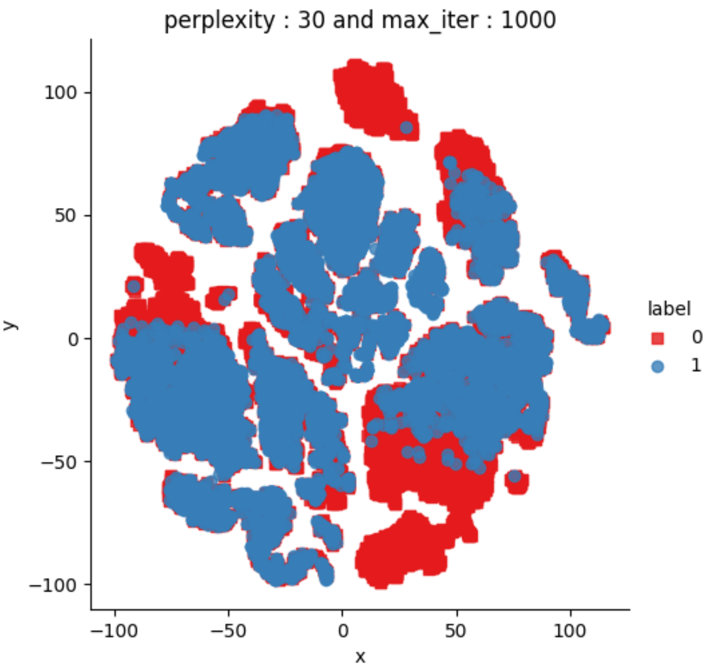
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 2 dimension

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'f
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 25000 samples in 0.083s...
    [t-SNE] Computed neighbors for 25000 samples in 4.626s...
    [t-SNE] Computed conditional probabilities for sample 1000 / 25000
    [t-SNE] Computed conditional probabilities for sample 2000 / 25000
    [t-SNE] Computed conditional probabilities for sample 3000 / 25000
    [t-SNE] Computed conditional probabilities for sample 4000 / 25000
    [t-SNE] Computed conditional probabilities for sample 5000 / 25000
    [t-SNE] Computed conditional probabilities for sample 6000 / 25000
    [t-SNE] Computed conditional probabilities for sample 7000 / 25000
    [t-SNE] Computed conditional probabilities for sample 8000 / 25000
    [t-SNE] Computed conditional probabilities for sample 9000 / 25000
    [t-SNE] Computed conditional probabilities for sample 10000 / 25000
    [t-SNE] Computed conditional probabilities for sample 11000 / 25000
    [t-SNE] Computed conditional probabilities for sample 12000 / 25000
    [t-SNE] Computed conditional probabilities for sample 13000 / 25000
    [t-SNE] Computed conditional probabilities for sample 14000 / 25000
    [t-SNE] Computed conditional probabilities for sample 15000 / 25000
    [t-SNE] Computed conditional probabilities for sample 16000 / 25000
    [t-SNE] Computed conditional probabilities for sample 17000 / 25000
    [t-SNE] Computed conditional probabilities for sample 18000 / 25000
    [t-SNE] Computed conditional probabilities for sample 19000 / 25000
    [t-SNE] Computed conditional probabilities for sample 20000 / 25000
    [t-SNE] Computed conditional probabilities for sample 21000 / 25000
    [t-SNE] Computed conditional probabilities for sample 22000 / 25000
    [t-SNE] Computed conditional probabilities for sample 23000 / 25000
    [t-SNE] Computed conditional probabilities for sample 24000 / 25000
    [t-SNE] Computed conditional probabilities for sample 25000 / 25000
    [t-SNE] Mean sigma: 0.083855
    [t-SNE] Computed conditional probabilities in 0.836s
    [t-SNE] Iteration 50: error = 107.9103699, gradient norm = 0.0222828 (50 iterations in 15.050s)
    [t-SNE] Iteration 100: error = 87.5088348, gradient norm = 0.0081066 (50 iterations in 11.618s)
    [t-SNE] Iteration 150: error = 83.7003937, gradient norm = 0.0051399 (50 iterations in 11.748s)
    [t-SNE] Iteration 200: error = 81.8966980, gradient norm = 0.0034294 (50 iterations in 11.419s)
    [t-SNE] Iteration 250: error = 80.8616486, gradient norm = 0.0025747 (50 iterations in 11.296s)
    [t-SNE] KL divergence after 250 iterations with early exaggeration: 80.861649
    [t-SNE] Iteration 300: error = 3.2320979, gradient norm = 0.0079634 (50 iterations in 11.485s)
    [t-SNE] Iteration 350: error = 2.6365247, gradient norm = 0.0078734 (50 iterations in 10.991s)
    [t-SNE] Iteration 400: error = 2.3160620, gradient norm = 0.0073537 (50 iterations in 10.976s)
    [t-SNE] Iteration 450: error = 2.1164856, gradient norm = 0.0068874 (50 iterations in 10.959s)
    [t-SNE] Iteration 500: error = 1.9801018, gradient norm = 0.0065020 (50 iterations in 10.911s)
    [t-SNE] Iteration 550: error = 1.8809927, gradient norm = 0.0061608 (50 iterations in 8.934s)
    [t-SNE] Iteration 600: error = 1.8061047, gradient norm = 0.0057827 (50 iterations in 10.989s)
    [t-SNE] Iteration 650: error = 1.7483189, gradient norm = 0.0053900 (50 iterations in 10.782s)
    [t-SNE] Iteration 700: error = 1.7028171, gradient norm = 0.0049884 (50 iterations in 10.936s)
    [t-SNE] Iteration 750: error = 1.6663076, gradient norm = 0.0046300 (50 iterations in 9.003s)
    [t-SNE] Iteration 800: error = 1.6364744, gradient norm = 0.0043076 (50 iterations in 11.582s)
    [t-SNE] Iteration 850: error = 1.6118417, gradient norm = 0.0039981 (50 iterations in 11.013s)
    [t-SNE] Iteration 900: error = 1.5911460, gradient norm = 0.0036676 (50 iterations in 15.100s)
    [t-SNE] Iteration 950: error = 1.5736196, gradient norm = 0.0033797 (50 iterations in 21.098s)
    [t-SNE] Iteration 1000: error = 1.5588248, gradient norm = 0.0031137 (50 iterations in 10.132s)
    [t-SNE] KL divergence after 1000 iterations: 1.558825
```

```
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,palette="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



3.6 Featurizing text data with tfidf weighted word-vectors

```
# avoid decoding problems
df = pd.read_csv("/content/drive/MyDrive/train.csv",nrows=25000)
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
df.head()
```

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

```
print("No of data points and features in data is:",df.shape)
```

```
    No of data points and features in data is: (25000, 6)
```

```
# 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_out(), tfidf.idf_))
```

- After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores.
- here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usage/vectors-similarity
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

```
nlp = spacy.load('en_core_web_lg')
vecs1 = []
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    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:
            #print word
            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%|██████████| 25000/25000 [03:21<00:00, 123.95it/s]
```

```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 25000/25000 [03:19<00:00, 125.62it/s]
```

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

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

```
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)
```

```
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 : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 300
Number of features in question2 w2v  dataframe : 300
Number of features in final dataframe  : 629
```

```
# 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

```
data=pd.read_csv('final_features.csv')
data.head()
```

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	290_y	291_y
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	...	88.905254	-110.995448
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	...	85.450115	46.592627
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	...	88.069179	15.648976
3	3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	46.318631	-81.313723
4	4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	...	114.466324	-64.204638

5 rows × 629 columns

```
y=data['is_duplicate']
print("No of data points and features",y.shape)
y.head()
```

```
No of data points and features (25000,)
0    0
1    0
2    0
3    0
4    0
Name: is_duplicate, dtype: int64
```

```
data=data.drop(labels=['is_duplicate','Unnamed: 0'],axis=1)
data.head()
```

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	...	290_y	29
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	...	88.905254	-110.995
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	...	85.450115	46.592
2	2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	...	88.069179	15.648
3	3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	...	46.318631	-81.313
4	4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	...	114.466324	-64.204

5 rows × 627 columns

```
print("Number of data points and features",data.shape)
```

```
Number of data points and features (25000, 627)
```

4.3 Random train test split(70:30)

```
X_train,X_test, y_train, y_test = train_test_split(data, y, stratify=y, test_size=0.3)
print("Number of data points in train data :",X_train.shape)
print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (17500, 627)
Number of data points in test data : (7500, 627)
```

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

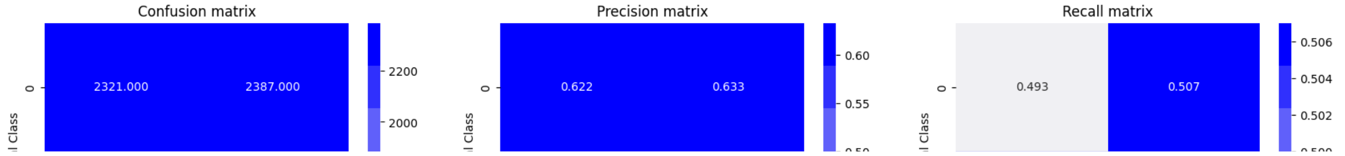
```
---------- Distribution of output variable in train data ----------
Class 0:  0.6277714285714285 Class 1:  0.3722285714285714
---------- Distribution of output variable in train data ----------
Class 0:  0.3722666666666667 Class 1:  0.3722666666666667
```

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

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # 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 diamensional array
    # 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 = [0,1]
    # 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()
```

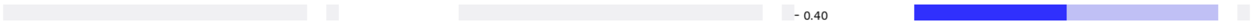
4.4 Building a random model (Finding worst-case log-loss)

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

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

```
Log loss on Test Data using Random Model 0.8957624514024181
```

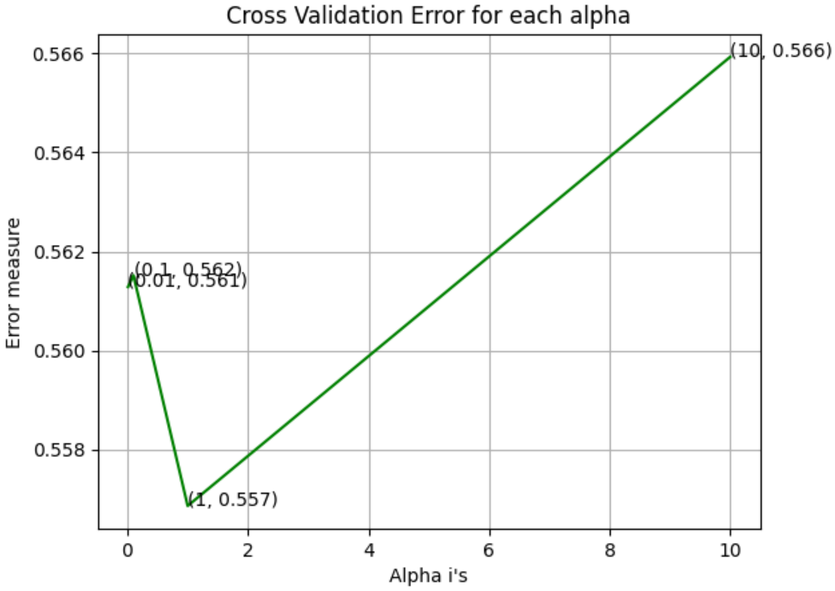


4.4 Logistic Regression with hyperparameter tuning



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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=’optimal’, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-
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 =  0.01 The log loss is: 0.5612821510110322
For values of alpha =  0.1 The log loss is: 0.5615186367701498
For values of alpha =  1 The log loss is: 0.5568561169839382
For values of alpha =  10 The log loss is: 0.565932702068257
```



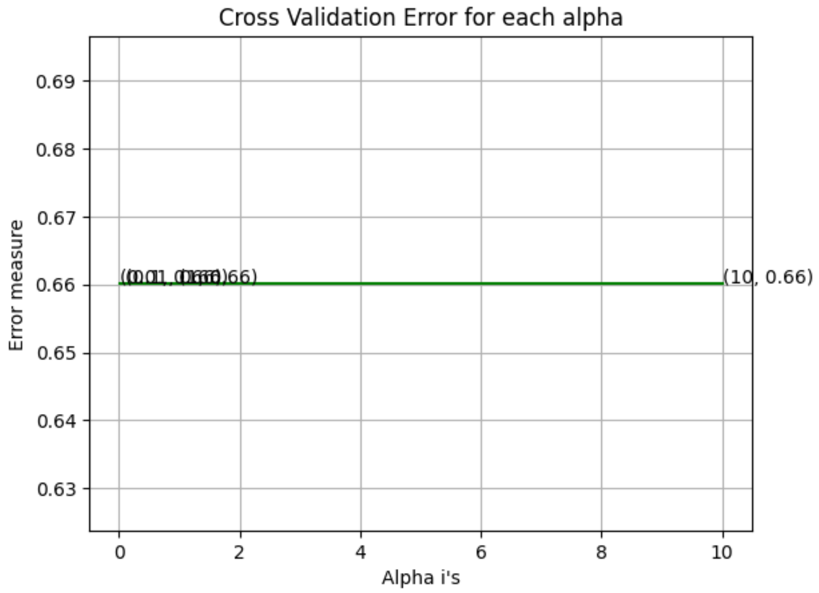
```
For values of best alpha =  1 The train log loss is: 0.5359292229710998
For values of best alpha =  1 The test log loss is: 0.5568561169839382
```

4.5 Linear SVM with hyperparameter tuning

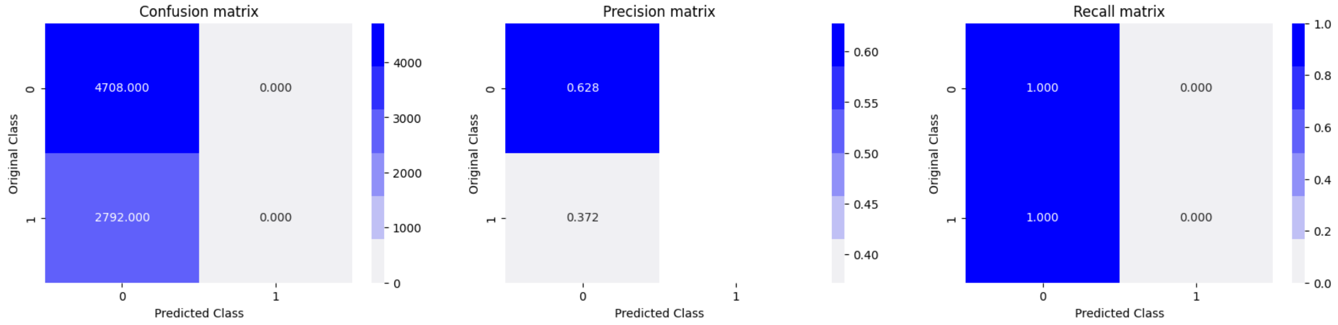
```
alpha = [10 ** x for x in range(-2, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=’optimal’, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1
predict_y = sig_clf.predict_proba(X_test)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-
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 =  0.01 The log loss is: 0.6601510300876712
    For values of alpha =  0.1 The log loss is: 0.6601510300876712
    For values of alpha =  1 The log loss is: 0.6601510300876712
    For values of alpha =  10 The log loss is: 0.6601510300876712
```



```
    For values of best alpha =  0.01 The train log loss is: 0.6601311308203897
    For values of best alpha =  0.01 The test log loss is: 0.6601510300876712
    Total number of data points : 7500
```



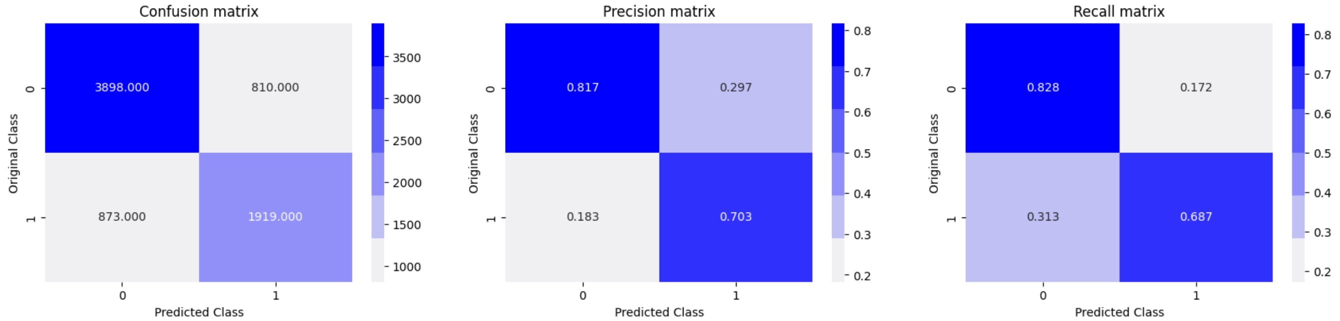
4.6 XGBoost

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
x_cfl=XGBClassifier()

prams={
    'n_estimators':[1,5,10],
    'max_depth':[3,5,10]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train,y_train)
print(random_cfl.best_params_)
```

```
    Fitting 5 folds for each of 9 candidates, totalling 45 fits
    {'n_estimators': 10, 'max_depth': 10}
```

```
result=random_cfl.best_params_
x_cfl=XGBClassifier(n_estimators=result['n_estimators'],max_depth=result['max_depth'])
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
predicted_y =np.argmax(predict_y,axis=1)
predicted_y =np.argmax(predict_y,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

```
train loss 0.3821891403407539
test loss 0.43436320003261014
```



✓ 7m 57s completed at 9:56 AM