Quora Question Pairs Similarity

1. Business Problem

1.1 Description

Quora is a place to gain and share knowledge—about anything. It's a platform to ask questions an insights and quality answers. This empowers people to learn from each other and to better unders

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

> Credits: Kaggle

__ Problem Statement __

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

1.2 Sources/Useful Links

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

__ Useful Links __

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/co
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-o

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.

2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold
3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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

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 dup

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

3. Exploratory Data Analysis and Feature Engineering

```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive',force_remount=True)
```

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

```
!pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/43/ff/74f23998ad2f93b94
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0
```

```
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
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import os
import gc
import re
from fuzzywuzzy import fuzz
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import warnings
import datetime as dt
from sklearn.decomposition import TruncatedSVD
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.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
import sys
from tqdm import tqdm
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
warnings.filterwarnings("ignore")
from sklearn.preprocessing import normalize
from sklearn.manifold import TSNE
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

3.1 Reading data and basic stats

```
!ls "/content/drive/My Drive/Colab Notebooks/"
```

```
 3_DonorsChoose_KNN.ipynb
 Clustering.ipynb
'Copy of parikshitgune@gmail.com_Assignment_6.ipynb'
 Dataset
 Decision_Tree.ipynb
 distance
 Keras_Mnist.ipynb
'parikshitgune@gmail (1).com_Assignment_3.ipynb'
 parikshitgune@gmail.com_Assignment_2.ipynb
 parikshitgune@gmail.com_Assignment_3.ipynb
 parikshitgune@gmail.com_Assignment_6.ipynb
 preprocessed_data.csv
 Quora_Case_Study
 Quora_Case_Study.ipynb
 temp.csv
 Untitled
 Untitled0.ipynb
'Untitled (1)'
 Untitled1.ipynb
```

```
df =  pd.read_csv('/content/drive/My Drive/Colab Notebooks/Quora_Case_Study/train.
```

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

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

```
df.head()
```

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

3.1.1 Very basic Data Description

```
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:

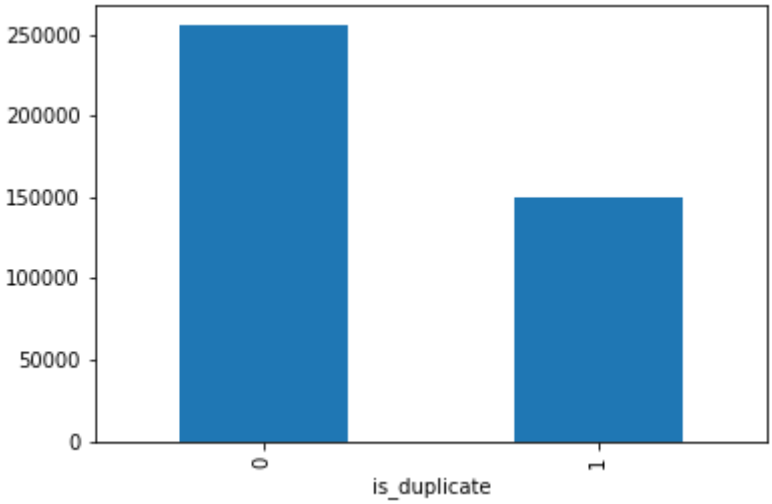
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicate

3.1.2 Distribution of data points among output classes

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1370232c18>
```




```
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 -
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(d
```

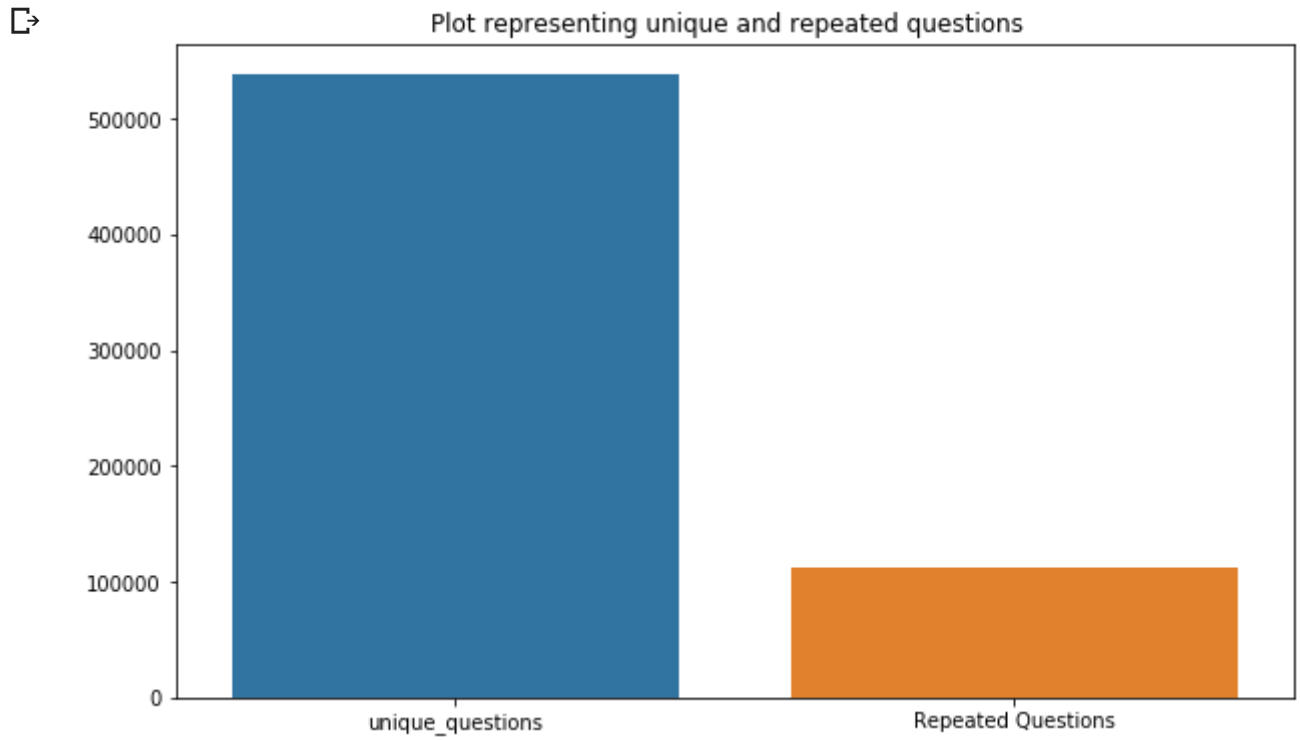
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.1.3 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'.for

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933
```

```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

plt.figure(figsize=(10, 6))
plt.title ("Plot representing unique and repeated questions  ")
sns.barplot(x,y)
plt.show()
```



3.1.4 Checking for Duplicates

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

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

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

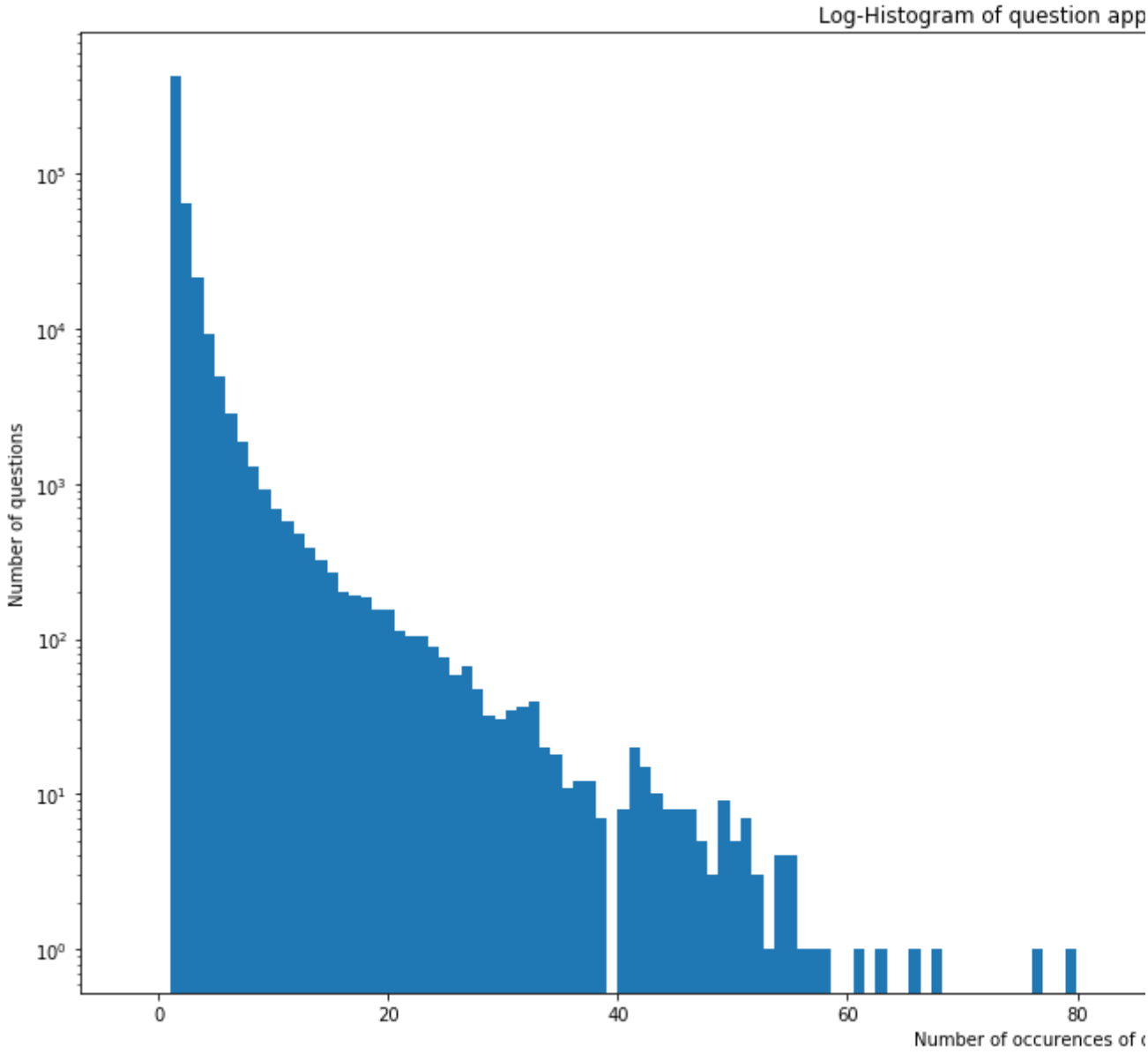
```
Number of duplicate questions 0
```

3.1.5 Number of occurrences of each question

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

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

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

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

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

```
plt.ylabel('Number of questions')

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

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



3.1.6 Checking for NULL values

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

```
            id  ...  is_duplicate
105780  105780  ...             0
201841  201841  ...             0
363362  363362  ...             0

[3 rows x 6 columns]
```

- There are two rows with null values in question2

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

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

3.2 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	q
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	
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	
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24} [/math] i...	0	1	1	
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	

3.2.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

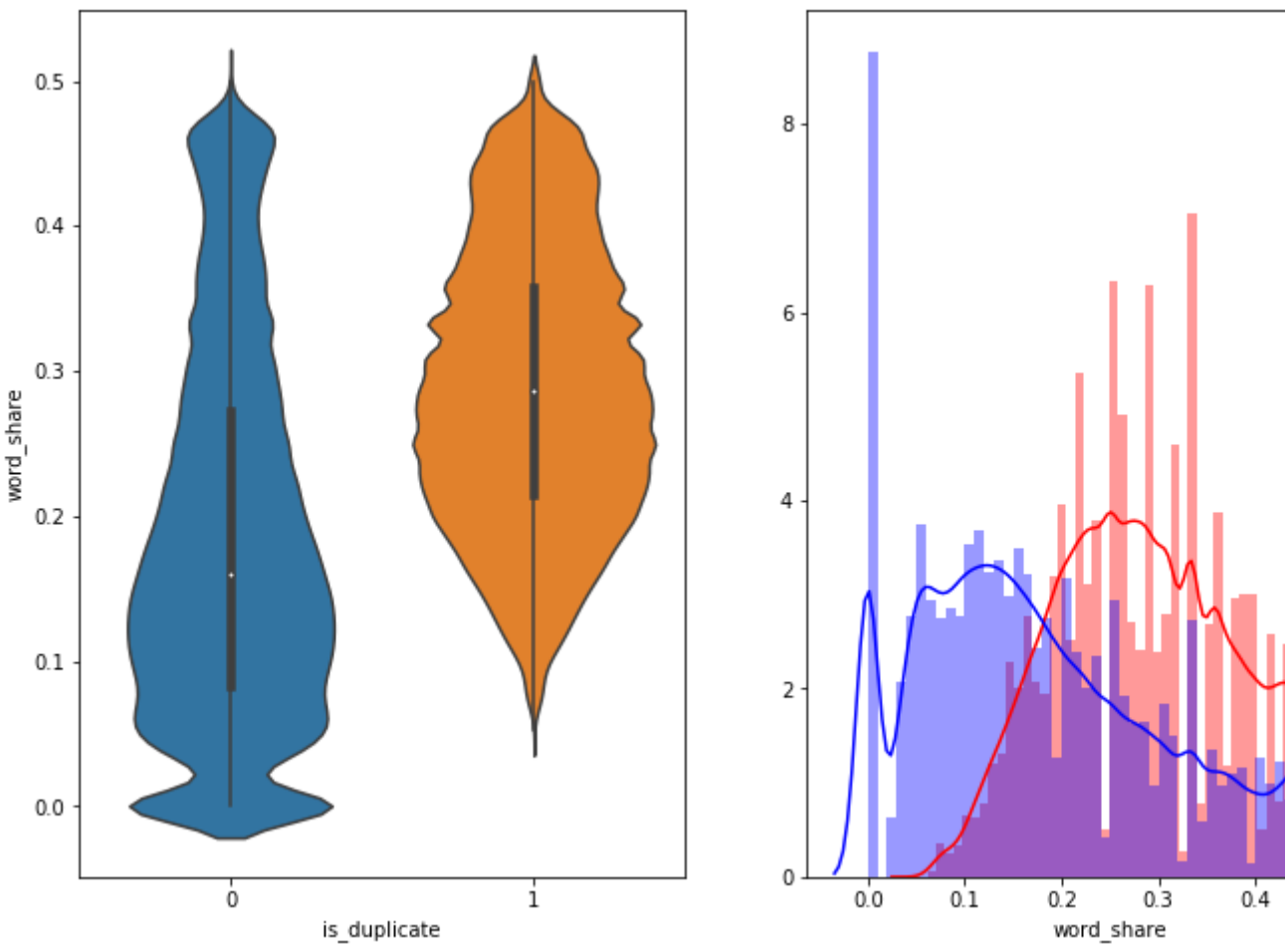
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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'
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words'
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.2.1.1 Feature: word_share

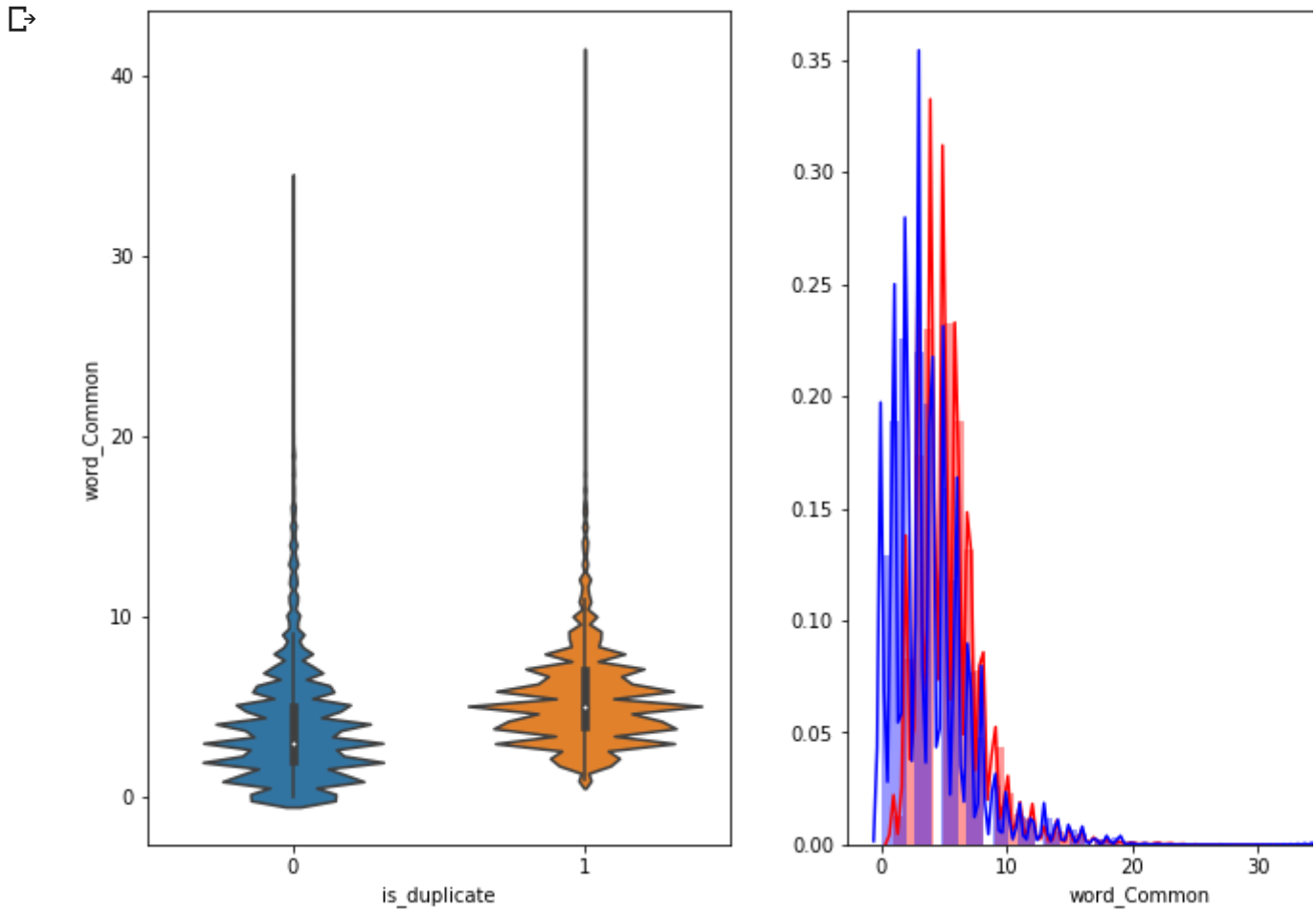
```
plt.figure(figsize=(12, 8))

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
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , color
plt.show()
```



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

3.2.1.2 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
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label = "0" , colo
plt.show()
```



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

3.3 Preprocessing of Text

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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
import nltk
nltk.download('stopwords')
```

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

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

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("′", "'").replace(
                           .replace("won't", "will not").replace("cannot", "can no
                           .replace("n't", " not").replace("what's", "what is").re
                           .replace("'ve", " have").replace("i'm", "i am").replace
                           .replace("he's", "he is").replace("she's", "she is").re
                           .replace("%", " percent ").replace("₹", " rupee ").repl
                           .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
```

3.4 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

- **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2
  cwc_min = common_word_count / (min(len(q1_words), len(q2_words))

- **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
  cwc_max = common_word_count / (max(len(q1_words), len(q2_words))

- **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2
  csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))

- **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    # Get the common non-stopwords from Question pair
```

```
        common_word_count = len(q1_words.intersection(q2_words))

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

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


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

        # 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 lcsubstrings(seq1, seq2, positions=False):
        from array import array
        L1, L2 = len(seq1), len(seq2)
        ms = []
        mlen = last = 0
        if L1 < L2:
          seq1, seq2 = seq2, seq1
          L1, L2 = L2, L1

        column = array('L', range(L2))

        for i in range(L1):
          for j in range(L2):
            old = column[j]
            if seq1[i] == seq2[j]:
              if i == 0 or j == 0:
                column[j] = 1
              else:
                column[j] = last + 1
              if column[j] > mlen:
                mlen = column[j]
                ms = [(i, j)]
              elif column[j] == mlen:
                ms.append((i, j))
            else:
              column[j] = 0
```

```
            last = old

        if positions:
          return (mlen, tuple((i - mlen + 1, j - mlen + 1) for i, j in ms if ms))
        return set(seq1[i - mlen + 1:i + 1] for i, _ in ms if ms)



    def get_longest_substr_ratio(a, b):
        strs = list(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["ques

        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-mat
        # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-function
        # https://github.com/seatgeek/fuzzywuzzy
        print("fuzzy features..")

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

        df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["q
        return df
```

```
extract_features(df)
```

```
token features...
fuzzy features..
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1

```
df.to_csv(r'/content/drive/My Drive/Colab Notebooks/preprocessed_data.csv')
```

step by step

3.4.1 Analysis of extracted features

3.4.1.1 Plotting Word clouds

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

increase

```
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
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).flat

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

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

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

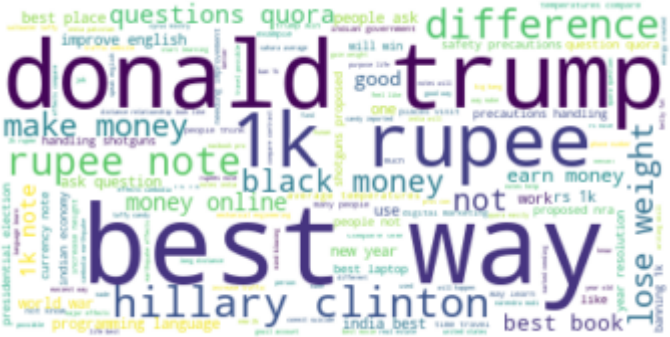
keywords

```
# 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(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

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

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

```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwor
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
```

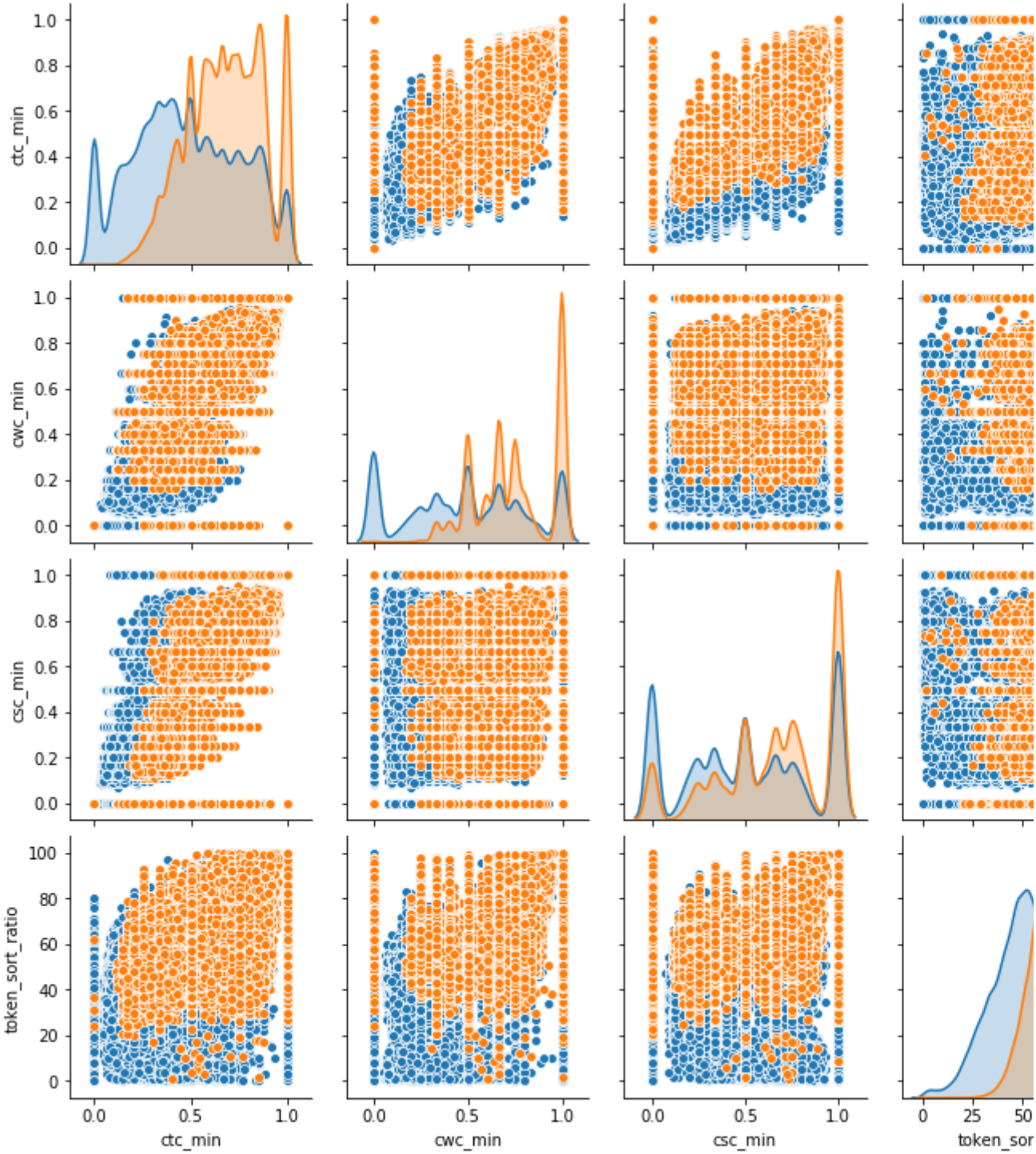



```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopword
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

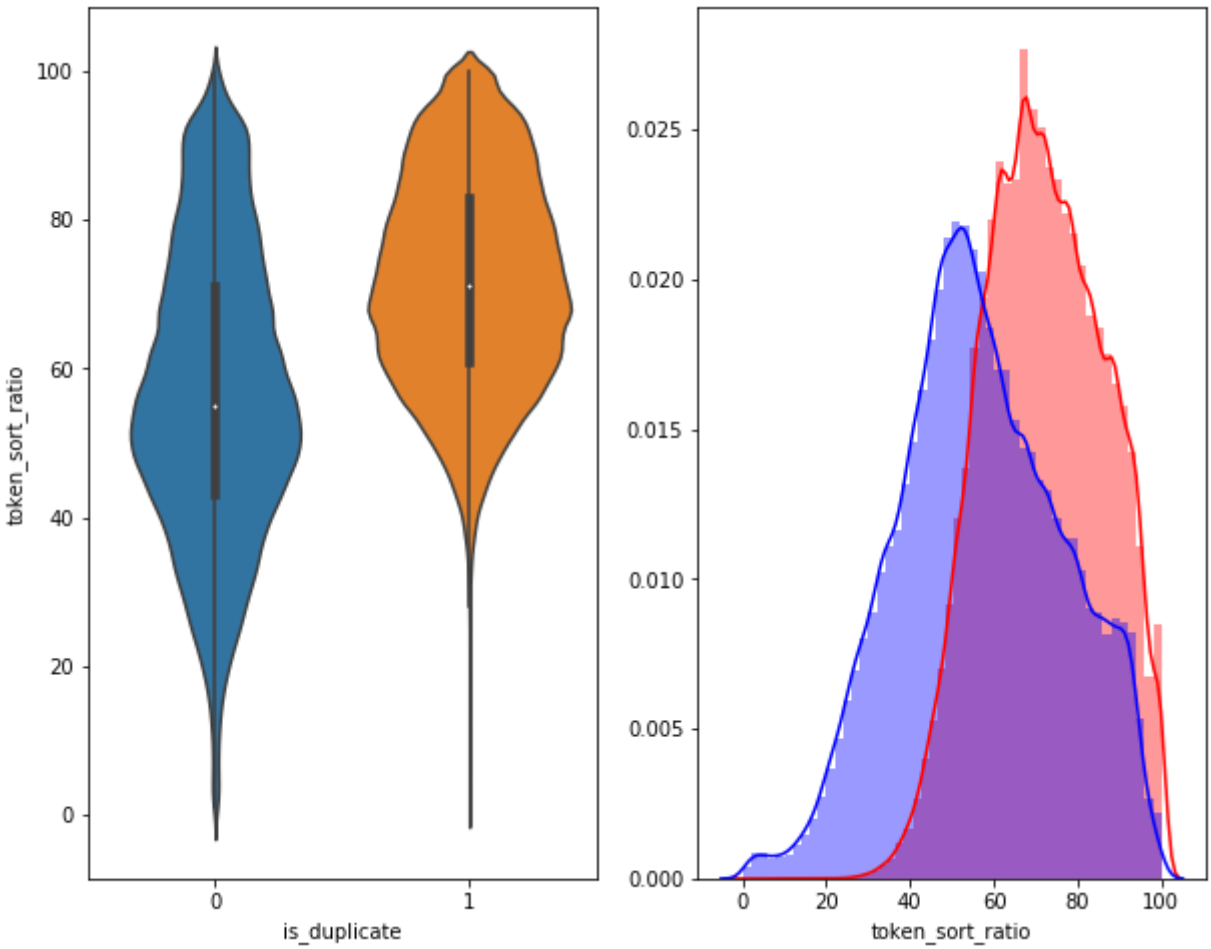
```
Word Cloud for non-Duplicate Question pairs:
```

3.4.1.2 Pair plot of features

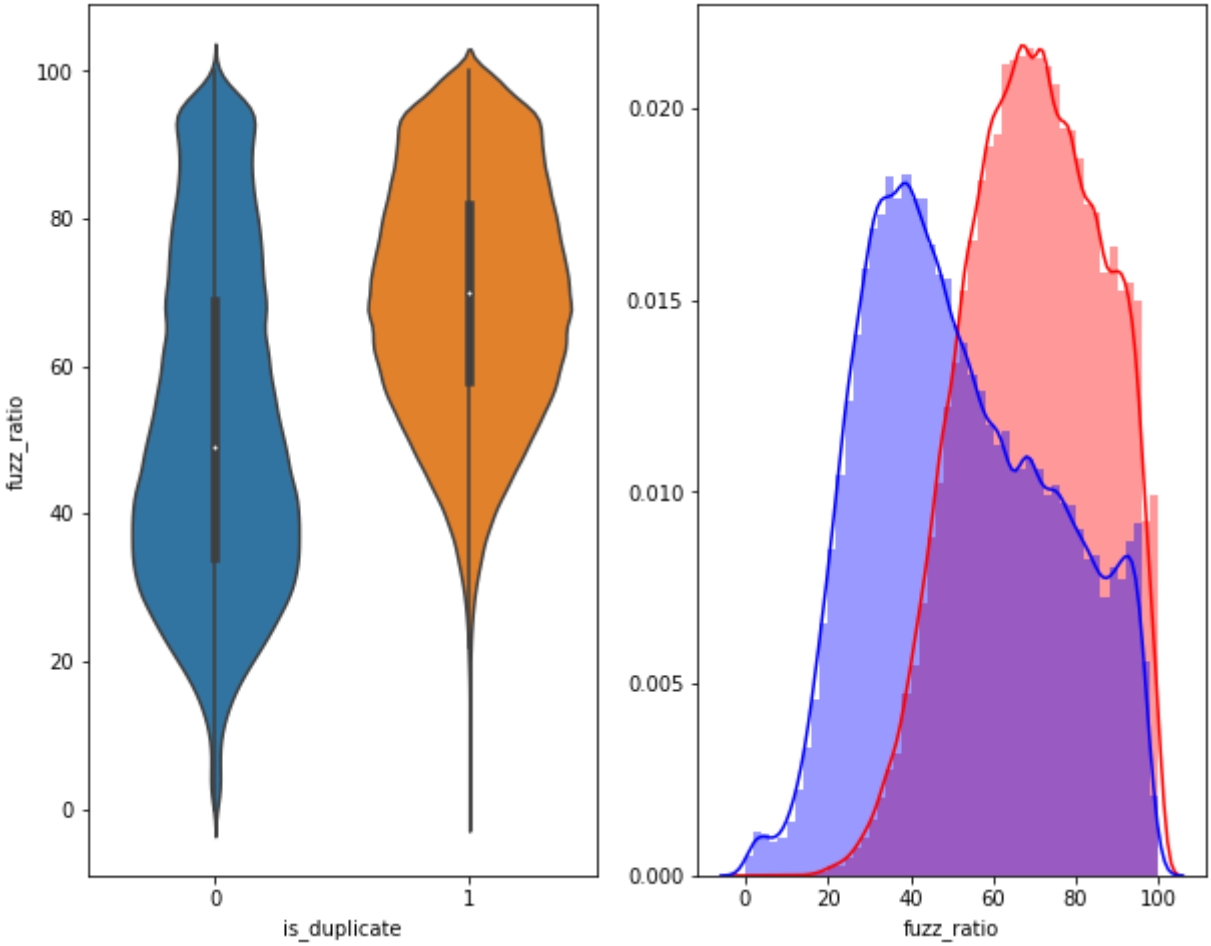
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicat
plt.show()
```




```
# 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",
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" ,
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
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , color
plt.show()
```

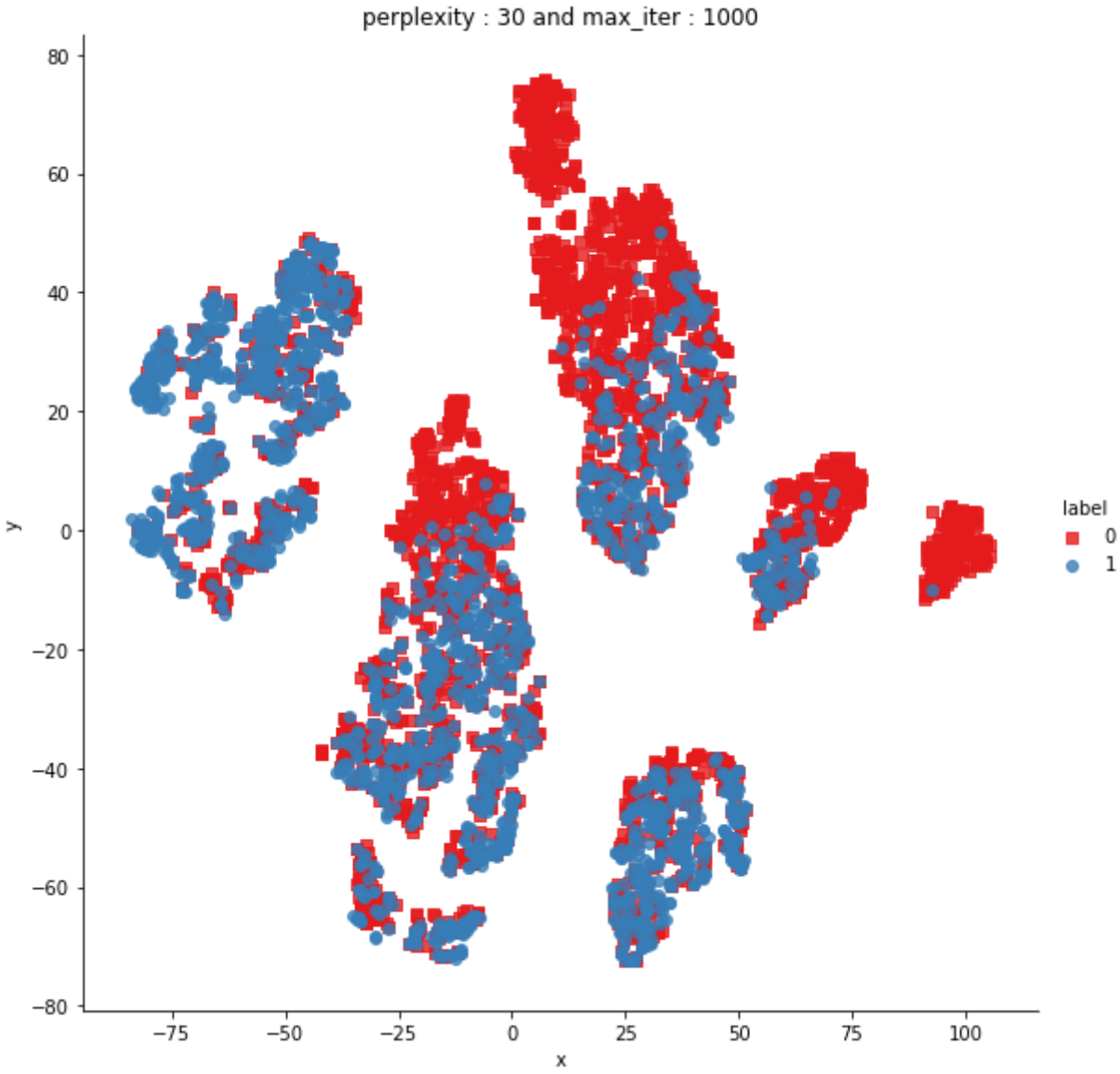


3.4.2 Visualization

```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min',
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 0.023s...
[t-SNE] Computed neighbors for 5000 samples in 0.424s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilities in 0.318s
[t-SNE] Iteration 50: error = 81.3425446, gradient norm = 0.0466835 (50 itera
[t-SNE] Iteration 100: error = 70.6490860, gradient norm = 0.0087385 (50 iter
[t-SNE] Iteration 150: error = 68.9494629, gradient norm = 0.0055224 (50 iter
[t-SNE] Iteration 200: error = 68.1286011, gradient norm = 0.0044136 (50 iter
[t-SNE] Iteration 250: error = 67.6222382, gradient norm = 0.0040027 (50 iter
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.622238
[t-SNE] Iteration 300: error = 1.7932034, gradient norm = 0.0011886 (50 itera
[t-SNE] Iteration 350: error = 1.3933792, gradient norm = 0.0004814 (50 itera
[t-SNE] Iteration 400: error = 1.2277224, gradient norm = 0.0002778 (50 itera
[t-SNE] Iteration 450: error = 1.1382111, gradient norm = 0.0001874 (50 itera
[t-SNE] Iteration 500: error = 1.0834070, gradient norm = 0.0001423 (50 itera
[t-SNE] Iteration 550: error = 1.0472494, gradient norm = 0.0001143 (50 itera
[t-SNE] Iteration 600: error = 1.0229402, gradient norm = 0.0000992 (50 itera
[t-SNE] Iteration 650: error = 1.0064085, gradient norm = 0.0000887 (50 itera
[t-SNE] Iteration 700: error = 0.9950162, gradient norm = 0.0000781 (50 itera
[t-SNE] Iteration 750: error = 0.9863962, gradient norm = 0.0000739 (50 itera
[t-SNE] Iteration 800: error = 0.9797970, gradient norm = 0.0000678 (50 itera
[t-SNE] Iteration 850: error = 0.9741811, gradient norm = 0.0000626 (50 itera
[t-SNE] Iteration 900: error = 0.9692637, gradient norm = 0.0000620 (50 itera
[t-SNE] Iteration 950: error = 0.9652759, gradient norm = 0.0000559 (50 itera
[t-SNE] Iteration 1000: error = 0.9615012, gradient norm = 0.0000559 (50 iter
[t-SNE] KL divergence after 1000 iterations: 0.961501
```

```
f1 = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

 draw the plot in appropriate place in the grid
ns.lmplot(data=df1, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1
lt.title("perplexity : {} and max_iter : {}".format(30, 1000))
lt.show()
```




```
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 0.020s...
[t-SNE] Computed neighbors for 5000 samples in 0.534s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilities in 0.313s
[t-SNE] Iteration 50: error = 80.5739822, gradient norm = 0.0296227 (50 itera
[t-SNE] Iteration 100: error = 69.4160385, gradient norm = 0.0032520 (50 iter
[t-SNE] Iteration 150: error = 68.0035553, gradient norm = 0.0018662 (50 iter
[t-SNE] Iteration 200: error = 67.4419785, gradient norm = 0.0012061 (50 iter
[t-SNE] Iteration 250: error = 67.1313705, gradient norm = 0.0008775 (50 iter
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.131371
[t-SNE] Iteration 300: error = 1.5172307, gradient norm = 0.0007258 (50 itera
[t-SNE] Iteration 350: error = 1.1812476, gradient norm = 0.0001984 (50 itera
[t-SNE] Iteration 400: error = 1.0386292, gradient norm = 0.0000930 (50 itera
[t-SNE] Iteration 450: error = 0.9660038, gradient norm = 0.0000607 (50 itera
[t-SNE] Iteration 500: error = 0.9280193, gradient norm = 0.0000515 (50 itera
[t-SNE] Iteration 550: error = 0.9082615, gradient norm = 0.0000439 (50 itera
[t-SNE] Iteration 600: error = 0.8948197, gradient norm = 0.0000341 (50 itera
[t-SNE] Iteration 650: error = 0.8839243, gradient norm = 0.0000353 (50 itera
[t-SNE] Iteration 700: error = 0.8753766, gradient norm = 0.0000331 (50 itera
[t-SNE] Iteration 750: error = 0.8696597, gradient norm = 0.0000279 (50 itera
[t-SNE] Iteration 800: error = 0.8648698, gradient norm = 0.0000248 (50 itera
[t-SNE] Iteration 850: error = 0.8604140, gradient norm = 0.0000254 (50 itera
[t-SNE] Iteration 900: error = 0.8561080, gradient norm = 0.0000236 (50 itera
[t-SNE] Iteration 950: error = 0.8519016, gradient norm = 0.0000246 (50 itera
[t-SNE] Iteration 1000: error = 0.8487377, gradient norm = 0.0000225 (50 iter
[t-SNE] KL divergence after 1000 iterations: 0.848738
```

```
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')
```

4. Machine Learning Models with TFIDF weighted W2V.

4.1 Featurizing text data with tfidf weighted word-vectors

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

import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

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

```
C:\Users\brahm\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: De
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
# avoid decoding problems
df =  pd.read_csv('/content/drive/My Drive/Colab Notebooks/Quora_Case_Study/train.

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


df.head()
```

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

```
# 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 ve
- here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usa
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

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

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)
```

```
df['q1_feats_m'] = list(vecs1)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
```

```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec1 = np.zeros([len(doc1), 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%|██████████████████████████████████████████████████████████████████████████████
```

```
#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 previo


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)


# dataframe of nlp features
df1.head()
```

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	

```
# data before preprocessing
df2.head()
```

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common
0	0	1	1	66	57	14	12	10.0
1	1	4	1	51	88	8	13	4.0
2	2	1	1	73	59	14	10	4.0
3	3	1	1	50	65	11	9	0.0
4	4	3	1	76	39	13	7	2.0

```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

	0	1	2	3	4	5	6	
0	121.929927	100.083900	72.497894	115.641800	-48.370870	34.619058	-172.057787	
1	-78.070939	54.843781	82.738482	98.191872	-51.234859	55.013510	-39.140730	
2	-5.355015	73.671810	14.376365	104.130241	1.433537	35.229116	-148.519385	
3	5.778359	-34.712038	48.999631	59.699204	40.661263	-41.658731	-36.808594	
4	51.138220	38.587312	123.639488	53.333041	-47.062739	37.356212	-298.722753	-

5 rows × 384 columns

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

	0	1	2	3	4	5	6	
0	125.983301	95.636485	42.114702	95.449980	-37.386295	39.400078	-148.116070	-8
1	-106.871904	80.290331	79.066297	59.302092	-42.175328	117.616655	-144.364237	-12
2	7.072875	15.513378	1.846914	85.937583	-33.808811	94.702337	-122.256856	-11
3	39.421531	44.136989	-24.010929	85.265863	-0.339022	-9.323137	-60.499651	-3
4	31.950101	62.854106	1.778164	36.218768	-45.130875	66.674880	-106.342341	-2

5 rows × 384 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
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 384
Number of features in question2 w2v  dataframe : 384
Number of features in final dataframe  : 794
```

```
# 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')
```

4.2 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 p

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

    # 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
    # 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 r
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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, yticklabe
    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, yticklabe
    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, yticklabe
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

4.2 Loading Data

4.2.1 Reading data from file and storing into sql table

```
#Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
```

```
        index_start = 1
        for df in pd.read_csv('final_features.csv', names=['Unnamed: 0','id','is_dupli
            df.index += index_start
            j+=1
            print('{} rows'.format(j*chunksize))
            df.to_sql('data', disk_engine, if_exists='append')
            index_start = df.index[-1] + 1


    #http://www.sqlitetutorial.net/sqlite-python/create-tables/
    def create_connection(db_file):
        """ create a database connection to the SQLite database
            specified by db_file
        :param db_file: database file
        :return: Connection object or None
        """
        try:
            conn = sqlite3.connect(db_file)
            return conn
        except Error as e:
            print(e)

        return None


    def checkTableExists(dbcon):
        cursr = dbcon.cursor()
        str = "select name from sqlite_master where type='table'"
        table_names = cursr.execute(str)
        print("Tables in the databse:")
        tables =table_names.fetchall()
        print(tables[0][0])
        return(len(tables))


    read_db = 'train.db'
    conn_r = create_connection(read_db)
    checkTableExists(conn_r)
    conn_r.close()
```

```
Tables in the databse:
data
```

```
    # try to sample data according to the computing power you have
    if os.path.isfile(read_db):
        conn_r = create_connection(read_db)
        if conn_r is not None:
            # for selecting first 1M rows
            # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)

            # for selecting random points
            data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 10000
            conn_r.commit()
            conn_r.close()


    # remove the first row
```

```
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace=True)
```

```
data.head()
```

	cwc_min	cwc_max	csc_min	csc_max	
1	0.199996000079998	0.166663888935184	0.0	0.0	0.1428
2	0.399992000159997	0.399992000159997	0.499987500312492	0.499987500312492	0.44443
3	0.833319444675922	0.714275510349852	0.999983333611106	0.857130612419823	0.68749
4	0.0	0.0	0.599988000239995	0.499991666805553	0.24999
5	0.749981250468738	0.749981250468738	0.499987500312492	0.499987500312492	0.62499

5 rows × 794 columns

4.2.2 Converting strings to numerics

```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
cols = list(data.columns)
for i in cols:
    data[i] = data[i].apply(pd.to_numeric)
    print(i)
```

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

4.2.3 Random train test split(70:30)

```
X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true,
```

```
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 : (70000, 794)
Number of data points in test data : (30000, 794)
```

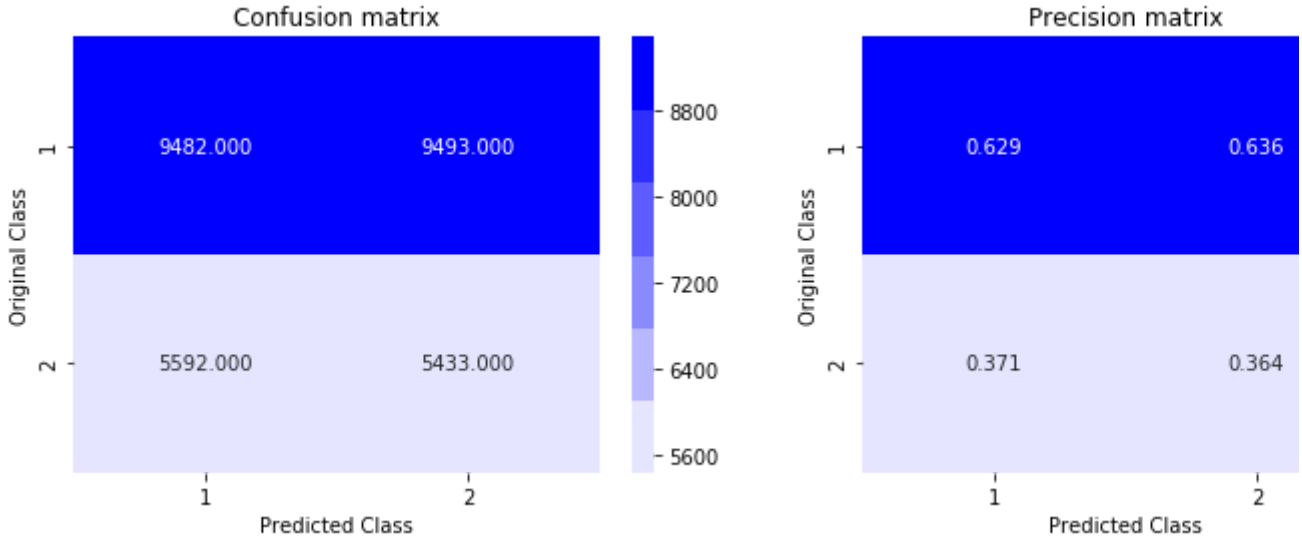
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/t
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)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

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6324857142857143 Class 1:  0.36751428571428574
---------- Distribution of output variable in train data ----------
Class 0:  0.3675 Class 1:  0.3675
```

4.3 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 su
# 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

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

```
Log loss on Test Data using Random Model 0.887242646958
```

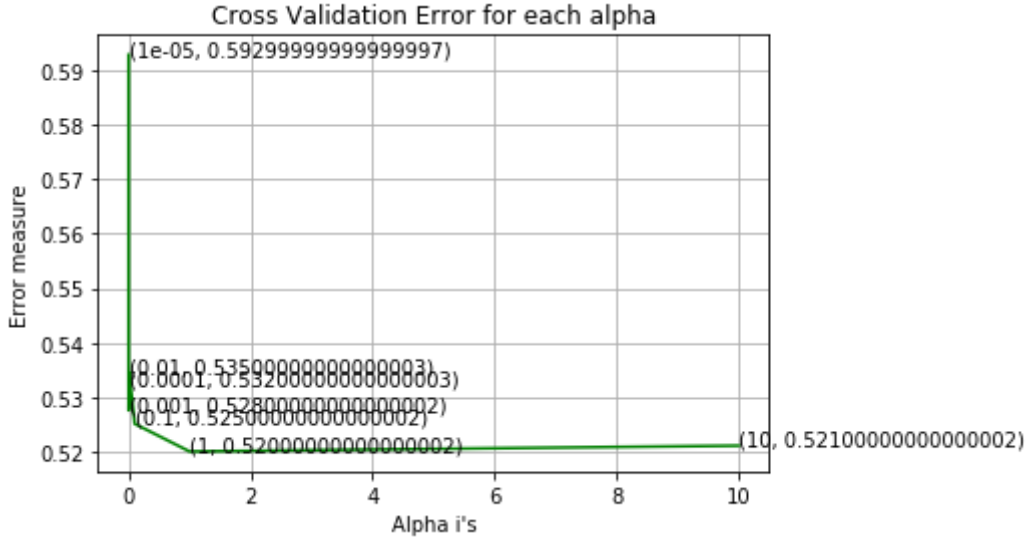


4.4 Logistic Regression with hyperparameter tuning

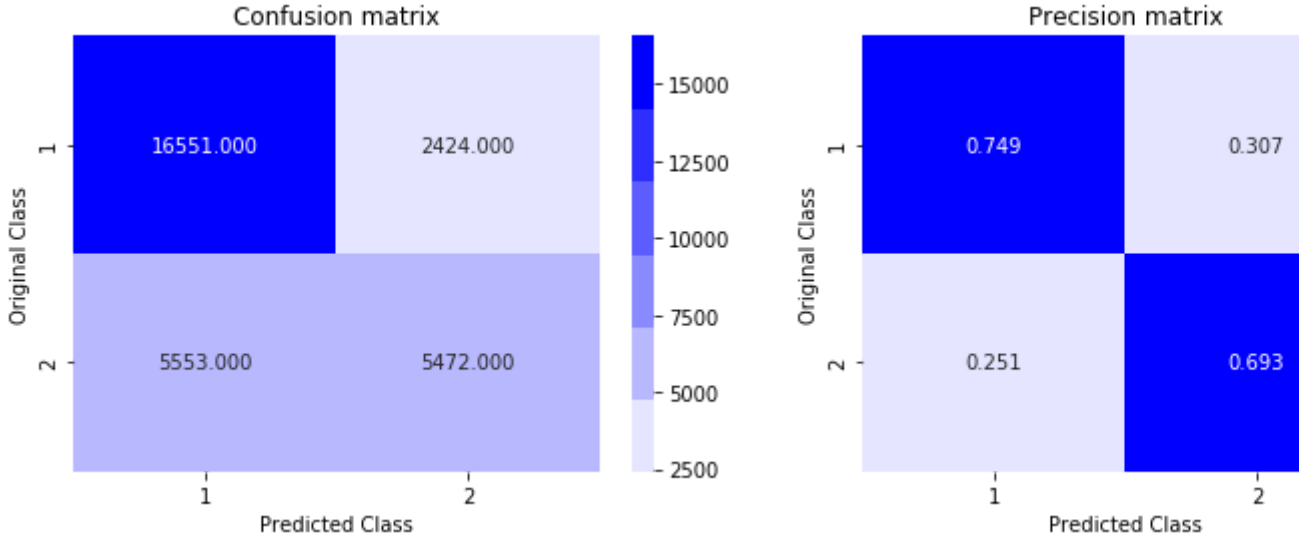
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

# some of methods
```

```
    # some of methods
    # fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradi
    # 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
        print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict

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

```
For values of alpha =  1e-05 The log loss is: 0.592800211149
For values of alpha =  0.0001 The log loss is: 0.532351700629
For values of alpha =  0.001 The log loss is: 0.527562275995
For values of alpha =  0.01 The log loss is: 0.534535408885
For values of alpha =  0.1 The log loss is: 0.525117052926
For values of alpha =  1 The log loss is: 0.520035530431
For values of alpha =  10 The log loss is: 0.521097925307
```




```
For values of best alpha =  1 The train log loss is: 0.513842874233
For values of best alpha =  1 The test log loss is: 0.520035530431
Total number of data points : 30000
```

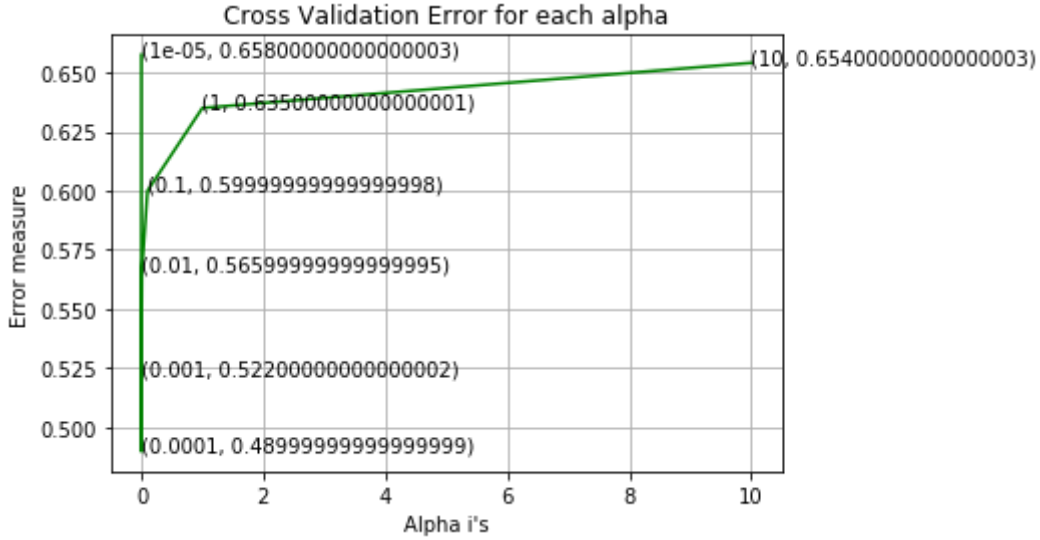


4.5 Linear SVM with hyperparameter tuning

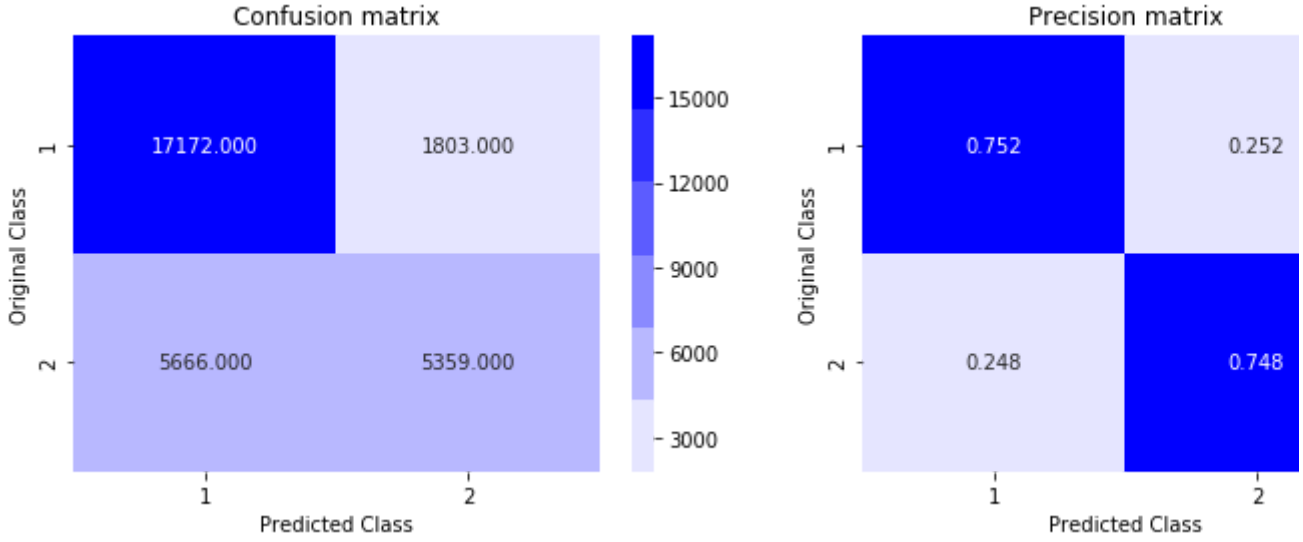
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inter
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
# 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 Gradi
# 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict

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

```
For values of alpha =  1e-05 The log loss is: 0.657611721261
For values of alpha =  0.0001 The log loss is: 0.489669093534
For values of alpha =  0.001 The log loss is: 0.521829068562
For values of alpha =  0.01 The log loss is: 0.566295616914
For values of alpha =  0.1 The log loss is: 0.599957866217
For values of alpha =  1 The log loss is: 0.635059427016
For values of alpha =  10 The log loss is: 0.654159467907
```




```
For values of best alpha =  0.0001 The train log loss is: 0.478054677285
For values of best alpha =  0.0001 The test log loss is: 0.489669093534
Total number of data points : 30000
```



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)
```

```
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

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
```

```
[0]     train-logloss:0.684819  valid-logloss:0.684845
Multiple eval metrics have been passed: 'valid-logloss' will be used for earl

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.61583   valid-logloss:0.616104
[20]    train-logloss:0.564616  valid-logloss:0.565273
[30]    train-logloss:0.525758  valid-logloss:0.52679
[40]    train-logloss:0.496661  valid-logloss:0.498021
[50]    train-logloss:0.473563  valid-logloss:0.475182
[60]    train-logloss:0.455315  valid-logloss:0.457186
[70]    train-logloss:0.440442  valid-logloss:0.442482
[80]    train-logloss:0.428424  valid-logloss:0.430795
[90]    train-logloss:0.418803  valid-logloss:0.421447
[100]   train-logloss:0.41069   valid-logloss:0.413583
[110]   train-logloss:0.403831  valid-logloss:0.40693
[120]   train-logloss:0.398076  valid-logloss:0.401402
[130]   train-logloss:0.393305  valid-logloss:0.396851
[140]   train-logloss:0.38913   valid-logloss:0.392952
[150]   train-logloss:0.385469  valid-logloss:0.389521
[160]   train-logloss:0.382327  valid-logloss:0.386667
[170]   train-logloss:0.379541  valid-logloss:0.384148
[180]   train-logloss:0.377014  valid-logloss:0.381932
[190]   train-logloss:0.374687  valid-logloss:0.379883
[200]   train-logloss:0.372585  valid-logloss:0.378068
[210]   train-logloss:0.370615  valid-logloss:0.376367
[220]   train-logloss:0.368559  valid-logloss:0.374595
[230]   train-logloss:0.366545  valid-logloss:0.372847
[240]   train-logloss:0.364708  valid-logloss:0.371311
[250]   train-logloss:0.363021  valid-logloss:0.369886
[260]   train-logloss:0.36144   valid-logloss:0.368673
[270]   train-logloss:0.359899  valid-logloss:0.367421
[280]   train-logloss:0.358465  valid-logloss:0.366395
[290]   train-logloss:0.357128  valid-logloss:0.365361
[300]   train-logloss:0.355716  valid-logloss:0.364315
[310]   train-logloss:0.354425  valid-logloss:0.363403
[320]   train-logloss:0.353276  valid-logloss:0.362595
[330]   train-logloss:0.352084  valid-logloss:0.361823
[340]   train-logloss:0.351051  valid-logloss:0.361167
[350]   train-logloss:0.349867  valid-logloss:0.36043
[360]   train-logloss:0.348829  valid-logloss:0.359773
[370]   train-logloss:0.347689  valid-logloss:0.359019
[380]   train-logloss:0.346607  valid-logloss:0.358311
[390]   train-logloss:0.345568  valid-logloss:0.357674
The test log loss is: 0.357054433715
```

5. Machine Learning Models with TFIDF Encoding

5.1 Featurizing text data with tfidf weighted word-vectors

```
df =  pd.read_csv('/content/drive/My Drive/Colab Notebooks/preprocessed_data.csv')
```

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

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

```
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(min_df = 10)
tfidf.fit(questions)

que1 = tfidf.transform(df['question1'])
que2 = tfidf.transform(df['question2'])
```

```
print("Question1 matrix :",que1.shape)
print("Question2 matrix :",que2.shape)
```

```
Question1 matrix : (404290, 20806)
Question2 matrix : (404290, 20806)
```

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

import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve


# remove the first row
df1 = df
df1 = df1.drop(['qid1','qid2','question1','question2'],axis=1)
target = df1['is_duplicate']
df1.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace=True)


df1.head()
```

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	wor
0	1	1	66	57	14	12	10.0	
1	4	1	51	88	8	13	4.0	
2	1	1	73	59	14	10	4.0	
3	1	1	50	65	11	9	0.0	
4	3	1	76	39	13	7	2.0	

5.1.1 Merging all data to form the final matrix

```
questions = hstack((que1,que2))
```

```
df1 = hstack((df1, questions),format="csr",dtype='float64')
df1.shape
```

```
(404290, 41638)
```

5.1.2 Train - Test split for ML Models

```
X_train,X_test, y_train, y_test = train_test_split(df1, target, stratify=target, t
print("train data :",x_train.shape)
print("test data :",x_test.shape)
```

```
train data : (283003, 41638)
test data : (121287, 41638)
```

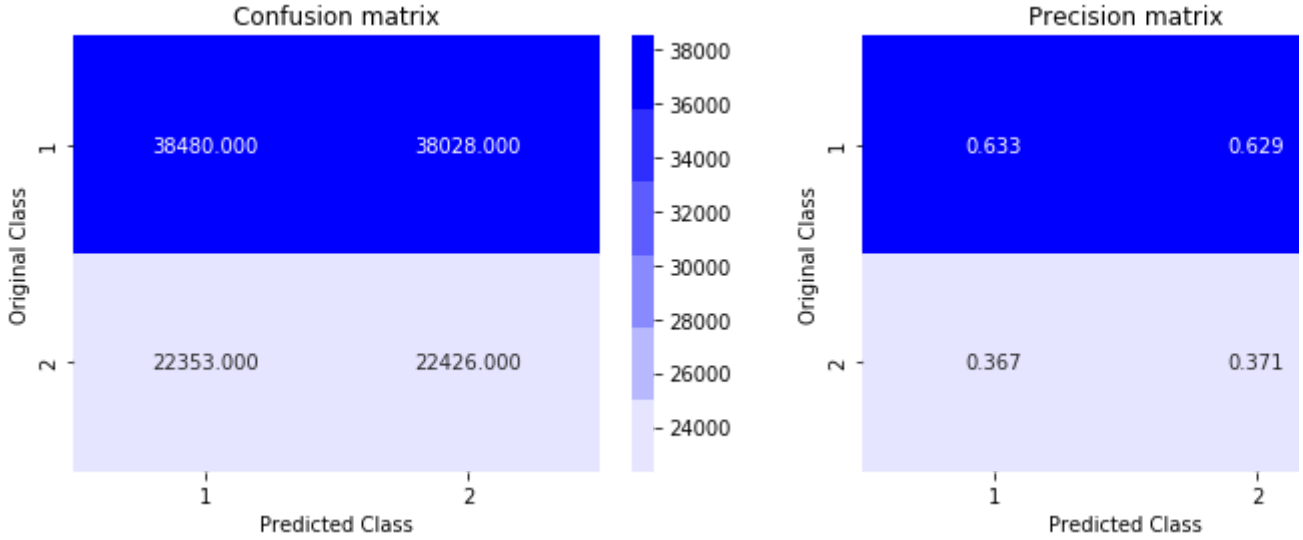
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/t
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)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

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in train data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

5.2 Random Model

```
# 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 su
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
test_len = len(y_test)
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

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

```
Log loss on Test Data using Random Model 0.8860096257134719
```

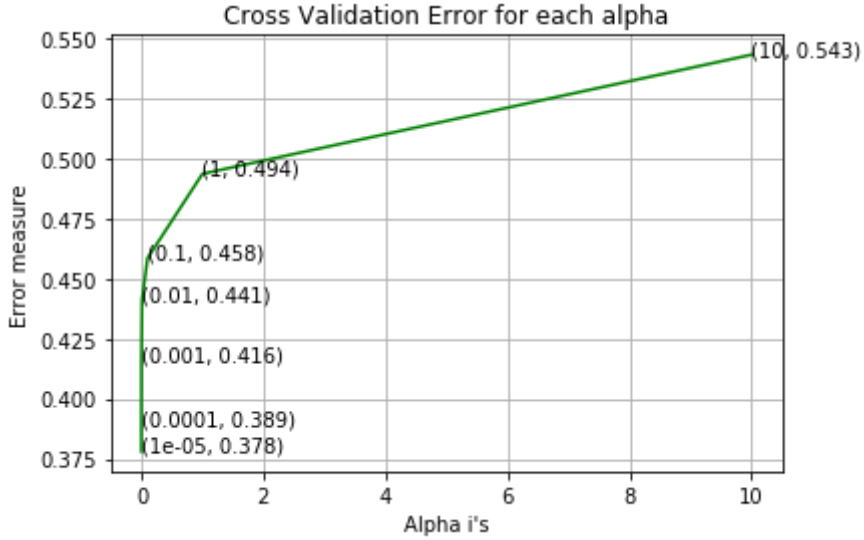


5.3 Logistic Regression with hyperparameter tuning

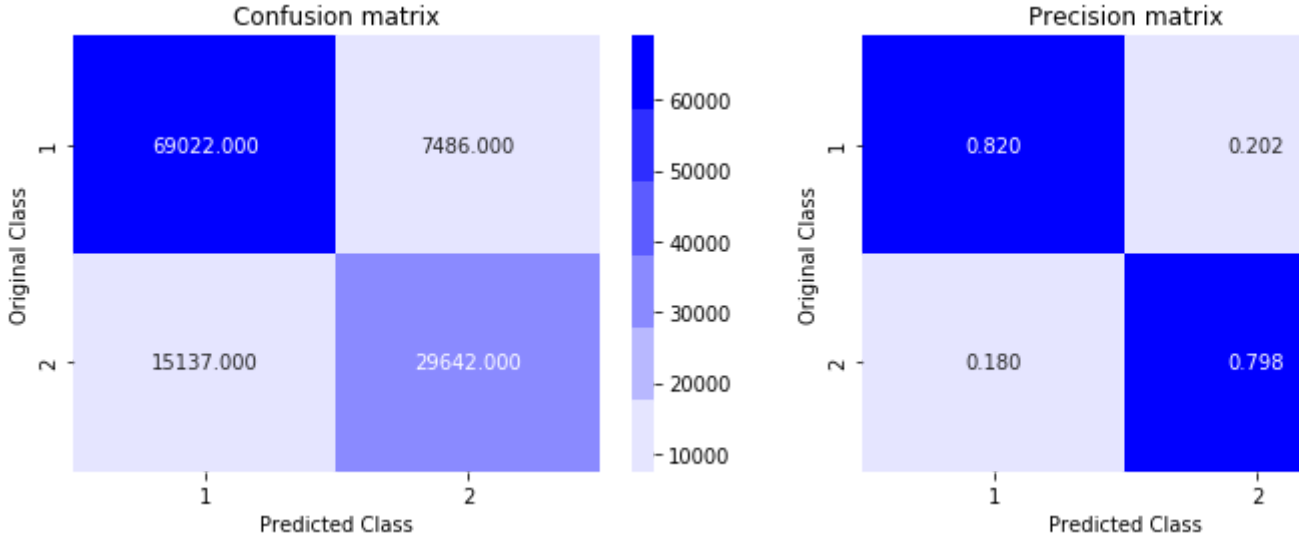
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inter
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
# 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 Gradi
# 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict

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

```
For values of alpha =  1e-05 The log loss is: 0.37800788686934805
For values of alpha =  0.0001 The log loss is: 0.3888833094007368
For values of alpha =  0.001 The log loss is: 0.4156489648488598
For values of alpha =  0.01 The log loss is: 0.4412279977292235
For values of alpha =  0.1 The log loss is: 0.45845706574855455
For values of alpha =  1 The log loss is: 0.493604158997163
For values of alpha =  10 The log loss is: 0.5432125457263784
```




```
For values of best alpha =  1e-05 The train log loss is: 0.373310031985806
For values of best alpha =  1e-05 The test log loss is: 0.37800788686934805
Total number of data points : 121287
```

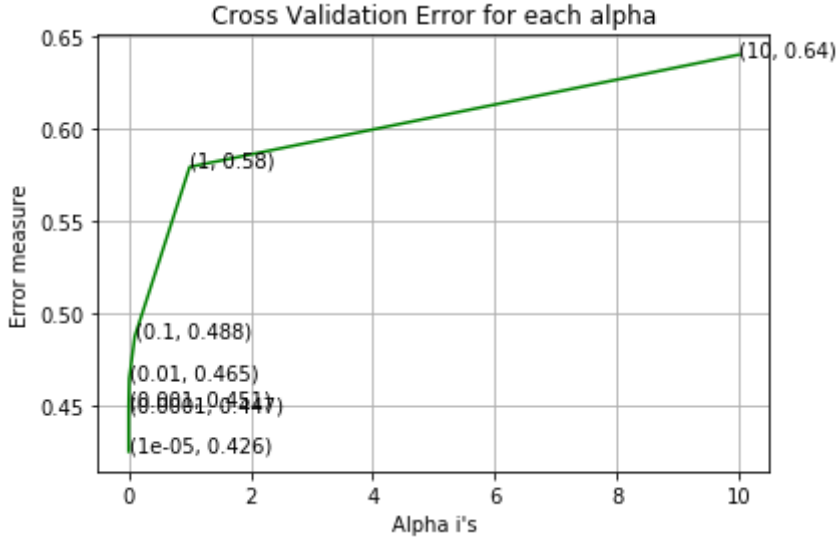


5.4 Linear SVM with hyperparameter tuning

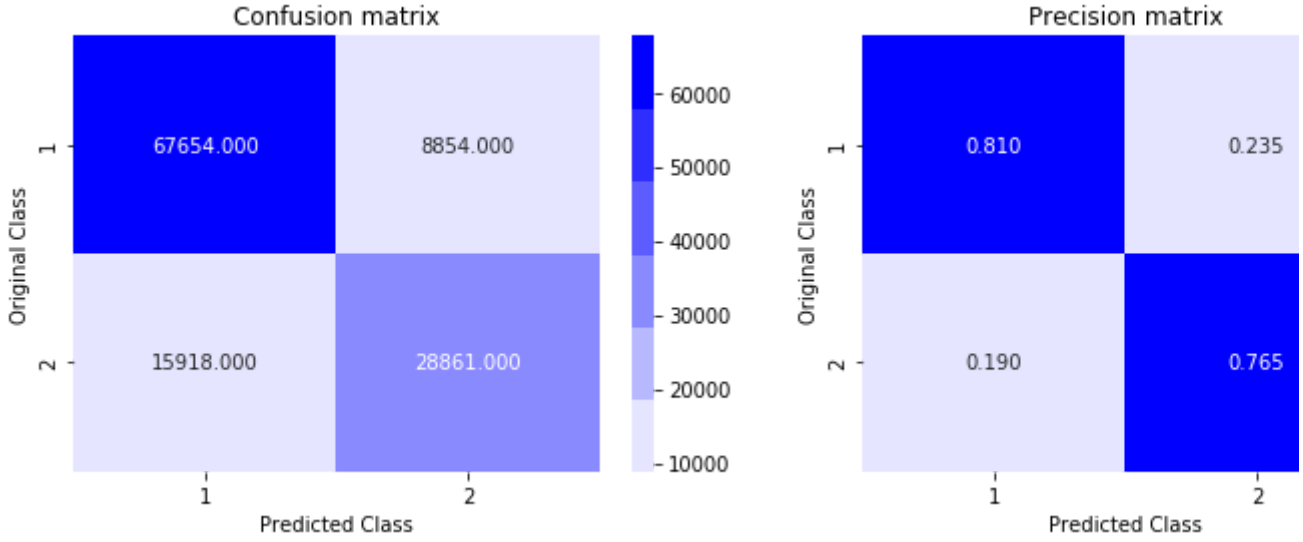
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inter
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate
# 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 Gradi
# 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict

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

```
For values of alpha =  1e-05 The log loss is: 0.4255580199572785
For values of alpha =  0.0001 The log loss is: 0.4473608994254777
For values of alpha =  0.001 The log loss is: 0.45137103562217257
For values of alpha =  0.01 The log loss is: 0.46502170550314414
For values of alpha =  0.1 The log loss is: 0.48784025936870445
For values of alpha =  1 The log loss is: 0.5796574083516003
For values of alpha =  10 The log loss is: 0.6404339644485744
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4233274932144767
For values of best alpha =  1e-05 The test log loss is: 0.4255580199572785
Total number of data points : 121287
```



5.5 XGBoost Model

5.5.1 Sampling and Splitting data

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

```
# taking first 100000 points
```

```
# taking 1 lakh data points

df2 = df1[:100000]
target2 = target[:100000]


print("total data: ",df2.shape)
print("total target data ",len(target2))
```

```
total data:  (100000, 41638)
total target data  100000
```

```
# taking random
X_train, X_test, y_train, y_test = train_test_split(df2, target2, stratify=target2


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])/t
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
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6274571428571428 Class 1:  0.3725428571428571
---------- Distribution of output variable in train data ----------
Class 0:  0.3725333333333333 Class 1:  0.3725333333333333
```

5.5.2 Hyperparameter Tuning

```
n_estimators = [50, 250, 450, 650, 850, 1050, 1250, 1450]
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
col_sample = [0.1, 0.3, 0.5, 0.7 , 0.9, 1]
subsample = [0.1,0.3,0.5,0.7,0.9,1]
def hp_tuning(X,Y):
  param_grid = dict(learning_rate=learning_rate,n_estimators=n_estimators,col_samp
  model = XGBClassifier(nthread=-1)
  k_fold = StratifiedKFold(n_splits=4, shuffle=True)
  random_search = RandomizedSearchCV(model, param_grid, scoring="neg_log_loss", n_
  result = random_search.fit(X,Y)
  # Summarize results
  print("Best: %f using %s" % (result.best_score_, result.best_params_))
  print()
  means = result.cv_results_['mean_test_score']
  stds = result.cv_results_['std_test_score']
  params = result.cv_results_['params']
  for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
  return result


result = hp_tuning(X_train,y_train)
```

```
Best: -0.330899 using {'subsample': 1, 'n_estimators': 1250, 'learning_rate':

-0.417424 (0.001187) with: {'subsample': 1, 'n_estimators': 250, 'learning_ra
-0.332140 (0.003440) with: {'subsample': 0.5, 'n_estimators': 650, 'learning_
-0.683954 (0.000071) with: {'subsample': 1, 'n_estimators': 250, 'learning_ra
-0.342774 (0.004811) with: {'subsample': 0.3, 'n_estimators': 1050, 'learning
-0.356267 (0.002168) with: {'subsample': 0.7, 'n_estimators': 1450, 'learning
-0.388471 (0.001695) with: {'subsample': 1, 'n_estimators': 450, 'learning_ra
-0.355696 (0.002104) with: {'subsample': 0.9, 'n_estimators': 50, 'learning_r
-0.656805 (0.000488) with: {'subsample': 0.9, 'n_estimators': 1050, 'learning
-0.330899 (0.003093) with: {'subsample': 1, 'n_estimators': 1250, 'learning_r
-0.333567 (0.002670) with: {'subsample': 1, 'n_estimators': 450, 'learning_ra
```

5.5.3 Running XGB classifier

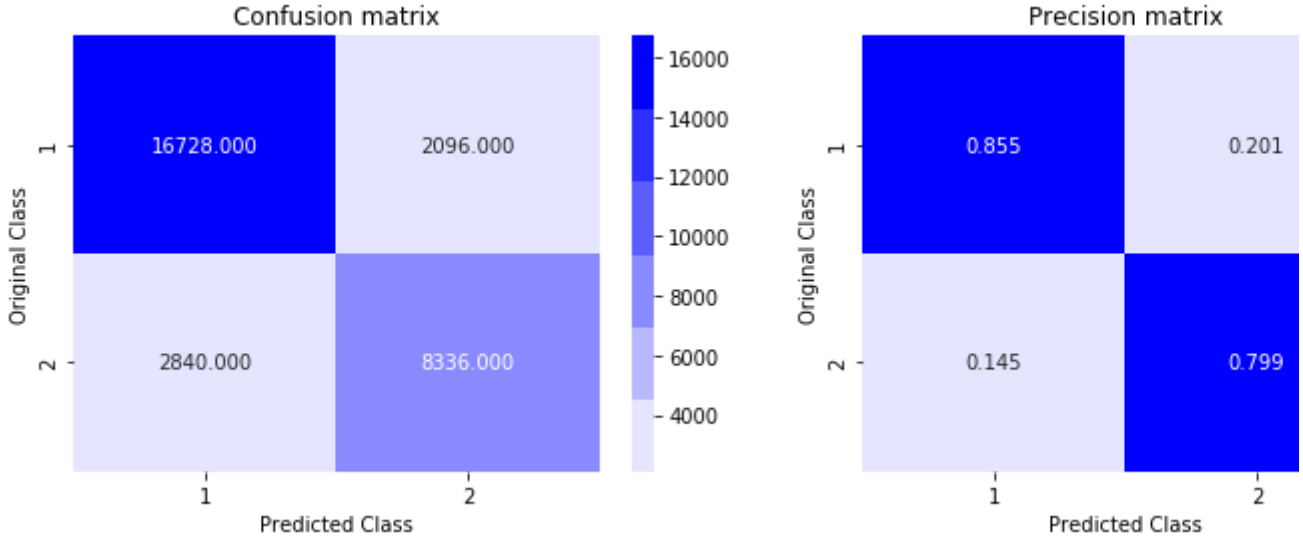
```
XGBClassifier(max_depth=4,learning_rate=0.1,n_estimators=1250,subsample=1,col_sam
```

```
import xgboost as XGBClassifier3

params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4
params['col_sample'] = 1
params['n_estimators'] = 1250
params['subsample'] = 1
params['learning_rate'] = 0.1
params['nthread'] = -1
params['silent'] = 1
d_train = XGBClassifier.DMatrix(X_train, label=y_train)
d_test = XGBClassifier.DMatrix(X_test, label=y_test)
watchlist = [(d_train, 'train'), (d_test, 'valid')]
bst = XGBClassifier.train(params, d_train, 400, watchlist,verbose_eval= 10,early_s
xgdmat = XGBClassifier.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
```

```
[0]     train-logloss:0.653484  valid-logloss:0.653293
Multiple eval metrics have been passed: 'valid-logloss' will be used for earl

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.465458  valid-logloss:0.465279
[20]    train-logloss:0.408995  valid-logloss:0.410054
[30]    train-logloss:0.385048  valid-logloss:0.386804
[40]    train-logloss:0.373542  valid-logloss:0.376315
[50]    train-logloss:0.365681  valid-logloss:0.369391
[60]    train-logloss:0.360497  valid-logloss:0.365223
[70]    train-logloss:0.356341  valid-logloss:0.361848
[80]    train-logloss:0.352495  valid-logloss:0.358819
[90]    train-logloss:0.349573  valid-logloss:0.356697
[100]   train-logloss:0.3469    valid-logloss:0.354905
[110]   train-logloss:0.343027  valid-logloss:0.35207
[120]   train-logloss:0.340577  valid-logloss:0.350256
[130]   train-logloss:0.338406  valid-logloss:0.348941
[140]   train-logloss:0.336754  valid-logloss:0.348012
[150]   train-logloss:0.334744  valid-logloss:0.346803
[160]   train-logloss:0.332688  valid-logloss:0.345518
[170]   train-logloss:0.331151  valid-logloss:0.34456
[180]   train-logloss:0.329754  valid-logloss:0.343843
[190]   train-logloss:0.328076  valid-logloss:0.342888
[200]   train-logloss:0.32628   valid-logloss:0.342068
[210]   train-logloss:0.324701  valid-logloss:0.341408
[220]   train-logloss:0.32362   valid-logloss:0.340945
[230]   train-logloss:0.321735  valid-logloss:0.339817
[240]   train-logloss:0.320687  valid-logloss:0.339584
[250]   train-logloss:0.319514  valid-logloss:0.339209
[260]   train-logloss:0.318244  valid-logloss:0.338518
[270]   train-logloss:0.317144  valid-logloss:0.338153
[280]   train-logloss:0.316154  valid-logloss:0.337845
[290]   train-logloss:0.315228  valid-logloss:0.337563
[300]   train-logloss:0.31332   valid-logloss:0.336387
[310]   train-logloss:0.312267  valid-logloss:0.335952
[320]   train-logloss:0.311431  valid-logloss:0.335783
[330]   train-logloss:0.31014   valid-logloss:0.335221
[340]   train-logloss:0.309316  valid-logloss:0.334975
[350]   train-logloss:0.308557  valid-logloss:0.334591
[360]   train-logloss:0.307377  valid-logloss:0.334315
[370]   train-logloss:0.306498  valid-logloss:0.334135
[380]   train-logloss:0.305629  valid-logloss:0.33389
[390]   train-logloss:0.304825  valid-logloss:0.333563
[399]   train-logloss:0.30413   valid-logloss:0.333427
The test log loss is: 0.3334322159703445
```

```
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 : 30000
```



6. Summary of the Case Study

```
from prettytable import PrettyTable
table = PrettyTable()

table.field_names = ['Model','Number of data points','Text Encoding','Hyperparamet
table.add_row(["Random","~400k","TFIDF Weighted W2V","No","0.887"])
table.add_row(["Logistic Regression","~400k","TFIDF Weighted W2V","Yes","0.520"])
table.add_row(["Linear SVM","~400k","TFIDF Weighted W2V","Yes","0.489"])
table.add_row(["XGBoost","~100k","TFIDF Weighted W2V","No","0.357"])
table.add_row(["\n","\n","\n","\n","\n"])
table.add_row(["Random","~400k","TFIDF","No","0.886"])
table.add_row(["Logistic Regression","~400k","TFIDF","Yes","0.378"])
table.add_row(["Linear SVM","~400k","TFIDF","Yes","0.425"])
table.add_row(["XGBoost","~100k","TFIDF","Yes","0.333"])
print(table)
```

```
+---------------------+-----------------------+--------------------+---------
|        Model        | Number of data points |   Text Encoding    | Hyperpar
+---------------------+-----------------------+--------------------+---------
|        Random       |         ~400k         | TFIDF Weighted W2V |
| Logistic Regression |         ~400k         | TFIDF Weighted W2V |
|      Linear SVM     |         ~400k         | TFIDF Weighted W2V |
|       XGBoost       |         ~100k         | TFIDF Weighted W2V |
|                     |                       |                    |
|                     |                       |                    |
|        Random       |         ~400k         |       TFIDF        |
| Logistic Regression |         ~400k         |       TFIDF        |
|      Linear SVM     |         ~400k         |       TFIDF        |
|       XGBoost       |         ~100k         |       TFIDF        |
+---------------------+-----------------------+--------------------+---------
```