krishnakarthik.g16@gmail.com_20

June 17, 2019

Quora Question Pairs

1. Business Problem

1.1 Description

Quora is a place to gain and share knowledge—about anything. It's a platform to ask questions and connect with people who contribute unique insights and quality answers. This empowers people to learn from each other and to better understand the world.

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

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 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

2.2 Mapping the real world problem to an ML problem

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

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

```
In [2]: !pip3 install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437aefa0d0a92ccfb38895d94
    || 184kB 2.8MB/s
Building wheels for collected packages: distance
  Building wheel for distance (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d0f12db1c66dbae9c5442b3
Successfully built distance
Installing collected packages: distance
Successfully installed distance-0.1.3
```

```
In [3]: !pip3 install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7eaa1beab2b9323073815da
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

```
In [97]: import warnings
         warnings.filterwarnings("ignore")
         import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         from subprocess import check_output
         %matplotlib inline
         import plotly.offline as py
         py.init_notebook_mode(connected=True)
         import plotly.graph_objs as go
         import plotly.tools as tls
         import os
         import gc

         import re
         from nltk.corpus import stopwords
         import distance
         from nltk.stem import PorterStemmer
         from bs4 import BeautifulSoup
         import re
         from nltk.corpus import stopwords
         # This package is used for finding longest common subsequence between two strings
         # you can write your own dp code for this
         import distance
         from nltk.stem import PorterStemmer
         from bs4 import BeautifulSoup
         from fuzzywuzzy import fuzz
```

```
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image


from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from datetime import datetime
```

3.1 Reading data and basic stats

```
In [5]: from google.colab import drive
        drive.mount('/content/drive')


        df = pd.read_csv("/content/drive/My Drive/Quora/train.csv")
        print("Number of data points:",df.shape[0])
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6b

Enter your authorization code:
··········
Mounted at /content/drive
Number of data points: 404290
```

```
In [6]: df.head()
```

```
Out[6]:    id  qid1  ...                                          question2 is_duplicate
        0   0     1  ...  What is the step by step guide to invest in sh...            0
        1   1     3  ...  What would happen if the Indian government sto...            0
        2   2     5  ...  How can Internet speed be increased by hacking...            0
        3   3     7  ...  Find the remainder when [math]23^{24}[/math] i...            0
        4   4     9  ...            Which fish would survive in salt water?            0

        [5 rows x 6 columns]
```

```
In [7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
```

```
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

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

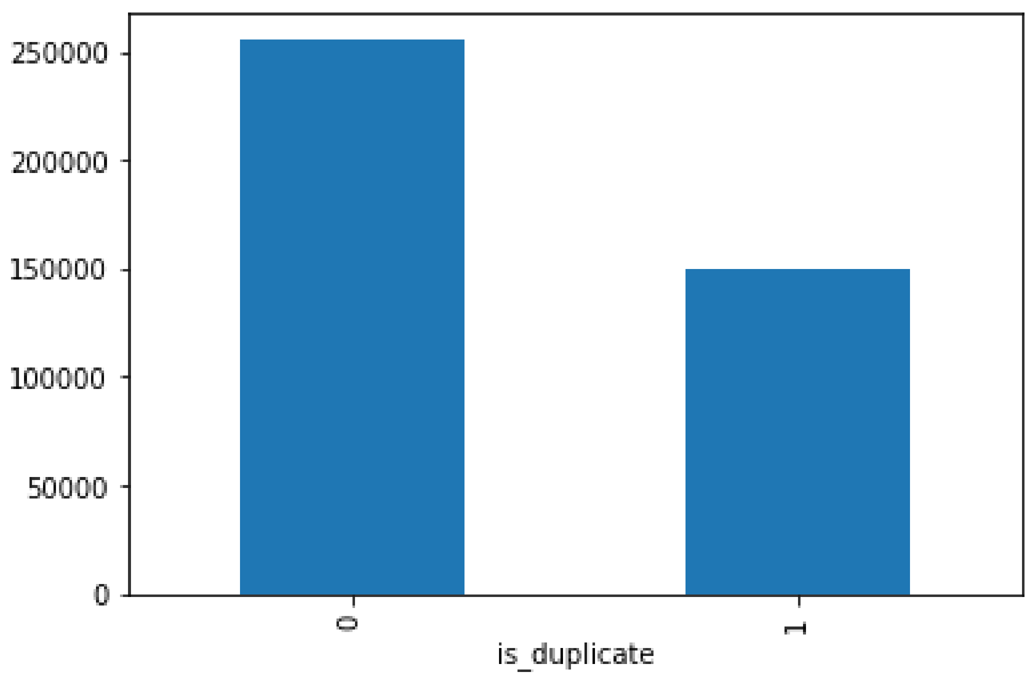
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

```
In [8]: df.groupby("is_duplicate")['id'].count().plot.bar()
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3bae51eef0>
```




```
In [9]: print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round
        print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_
```

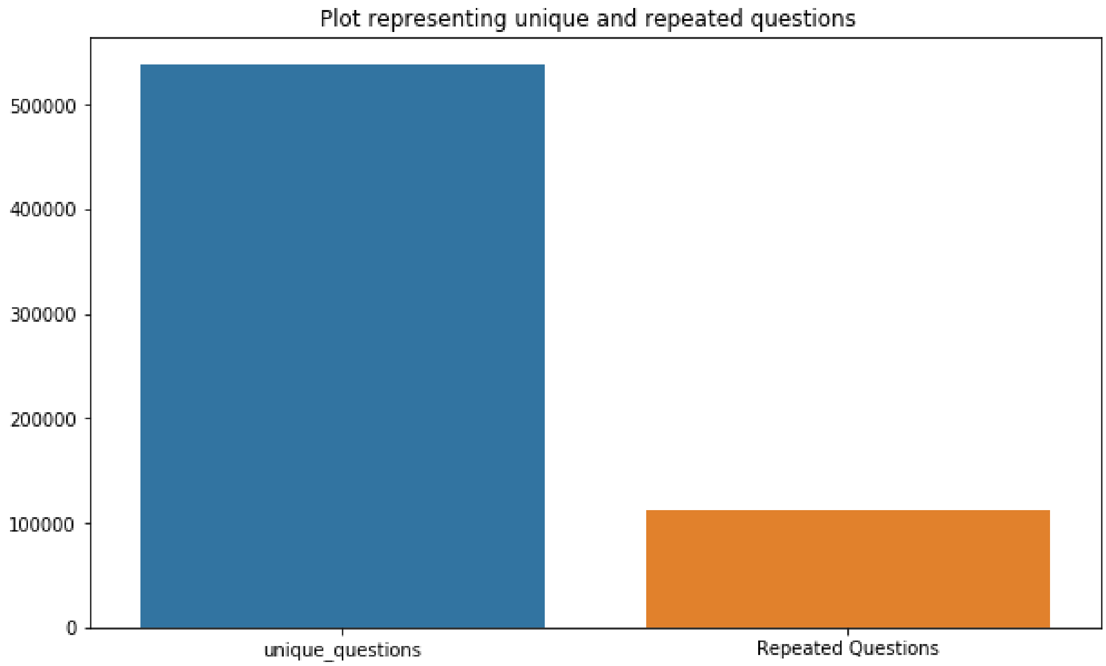
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.2 Number of unique questions

```
In [10]: 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(q

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

         q_vals=qids.value_counts()

         q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

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

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

```
In [11]: x = ["unique_questions" , "Repeated Questions"]
         y =  [unique_qs , qs_morethan_onetime]

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



3.2.3 Checking for Duplicates

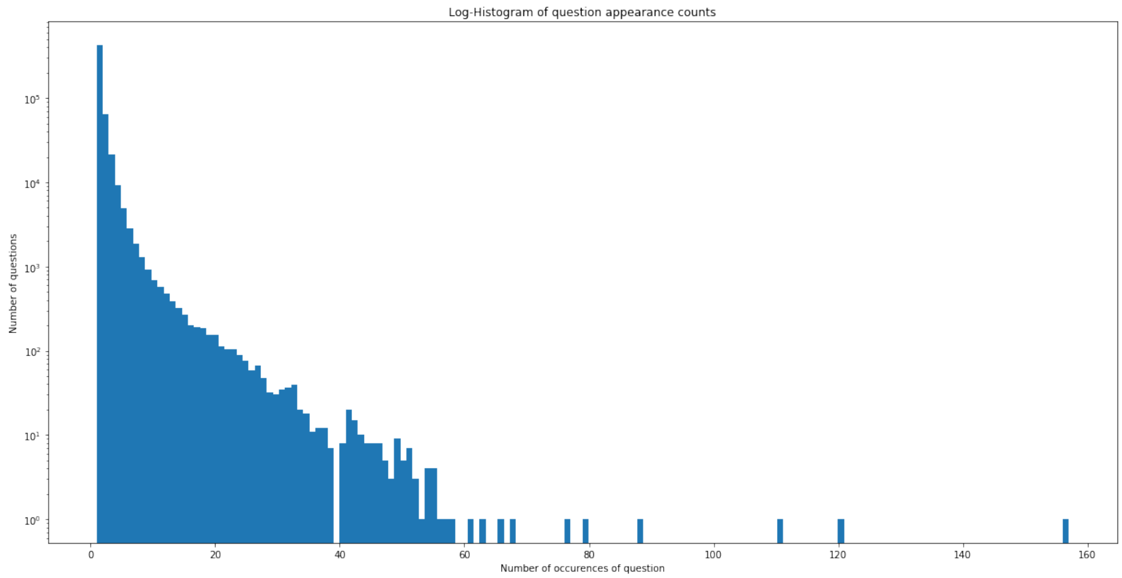
```
In [12]: #checking whether there are any repeated pair of questions

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

         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

```
In [13]: plt.figure(figsize=(20, 10))

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

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

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

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

         plt.ylabel('Number of questions')

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

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



3.2.5 Checking for NULL values

```
In [14]: #Checking whether there are any rows with null values
         nan_rows = df[df.isnull().any(1)]
         print (nan_rows)

         # Filling the null values with ' '
         df = df.fillna('')
         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]
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
```

0.1 3.2.6 Train-Test Split

```
In [15]: y = df['is_duplicate']
```

```
X_train,X_test, y_train, y_test = train_test_split(df, y, stratify=y, test_size=0.3)
print("Shape of train data",X_train.shape)
print("Shape of test data",X_test.shape)
```

```
Shape of train data (283003, 6)
Shape of test data (121287, 6)
```

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

```
In [0]: if os.path.isfile('df_fe_without_preprocessing_train_2.csv'):
            X_train = pd.read_csv("df_fe_without_preprocessing_train_2.csv",encoding='latin-1')

        else:
            X_train['freq_qid1'] = X_train.groupby('qid1')['qid1'].transform('count')
            X_train['freq_qid2'] = X_train.groupby('qid2')['qid2'].transform('count')

            X_train['q1len']      = X_train['question1'].str.len()
            X_train['q2len']      = X_train['question2'].str.len()
            X_train['q1_n_words'] = X_train['question1'].apply(lambda row: len(row.split(" ")))
            X_train['q2_n_words']      = X_train['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)
            X_train['word_Common'] = X_train.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))
            X_train['word_Total'] = X_train.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))
            X_train['word_share'] = X_train.apply(normalized_word_share, axis=1)
```

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

X_train.to_csv("df_fe_without_preprocessing_train_2.csv", index=False)
```

In [30]:
```
X_train.head()
```

Out[30]:
```
            id    qid1  ...  fuzz_partial_ratio longest_substr_ratio
111330  111330  182389  ...                  59             0.166667
253083  253083   42529  ...                  60             0.269231
137682  137682  138116  ...                  54             0.176471
374163  374163  505032  ...                  50             0.296296
307595  307595  431280  ...                  65             0.454545

[5 rows x 32 columns]
```

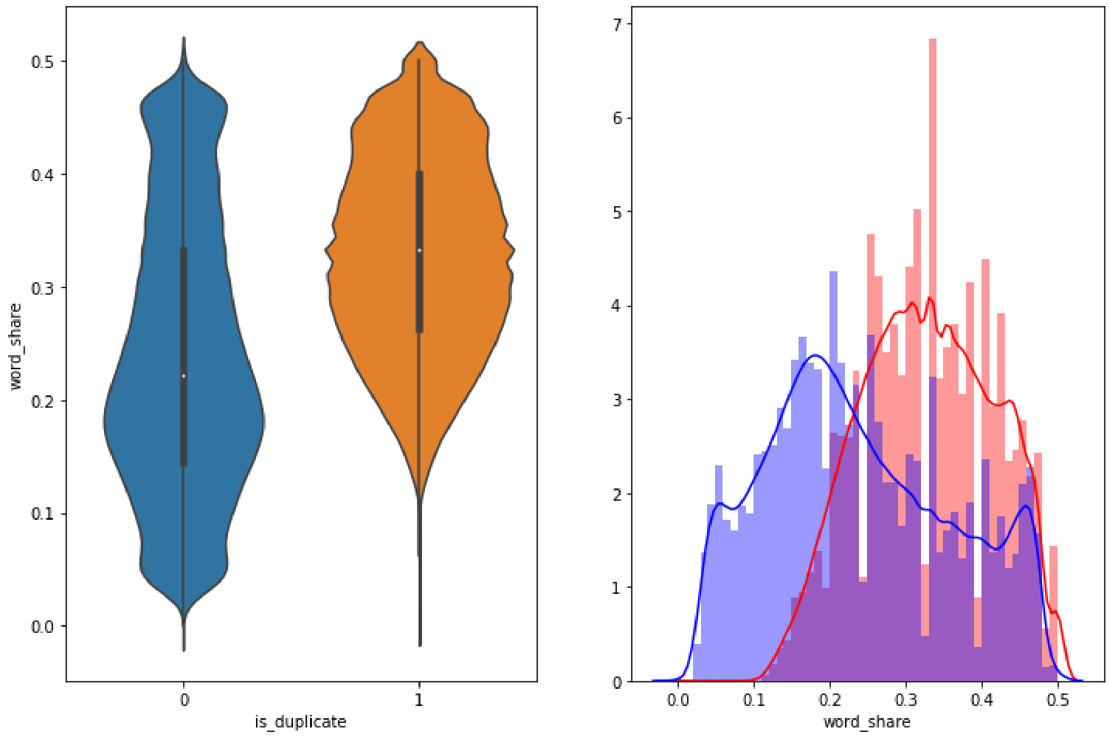
3.3.1 Analysis of some of the extracted features

In [31]:
```
print ("Minimum length of the questions in question1 : " , min(X_train['q1_n_words']))

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

print ("Number of Questions with minimum length [question1] :", X_train[X_train['q1_n_w
print ("Number of Questions with minimum length [question2] :", X_train[X_train['q2_n_w
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 34
Number of Questions with minimum length [question2] : 10
```

3.3.1.1 Feature: word_share

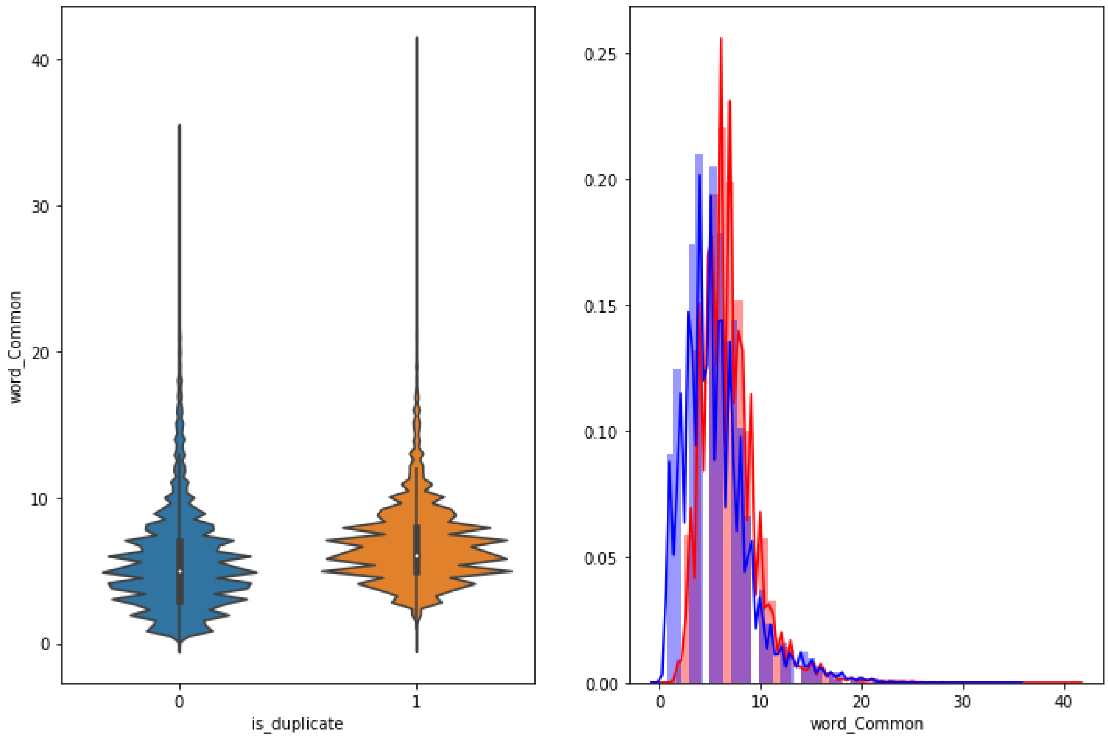
In [32]:
```
plt.figure(figsize=(12, 8))

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

plt.subplot(1,2,2)
sns.distplot(X_train[X_train['is_duplicate'] == 1.0]['word_share'][0:] , label = "1", c
sns.distplot(X_train[X_train['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" ,
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

In [33]:
```
plt.figure(figsize=(12, 8))

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

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



3.4 Preprocessing of Text

```
In [34]: # To get the results in 4 decemal points
         SAFE_DIV = 0.0001
         import nltk
         nltk.download('stopwords')
         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").r
                                     .replace("n't", " not").replace("what's", "what is").replace
                                     .replace("'ve", " have").replace("i'm", "i am").replace("'re
                                     .replace("he's", "he is").replace("she's", "she is").replace
                                     .replace("%", " percent ").replace("", " rupee ").replace("$
                                     .replace("", " euro ").replace("'ll", " will")
             x = re.sub(r"([0-9]+)000000", r"\1m", x)
             x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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


        return x
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

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

Features: - **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2 cwc_min = common_word_count / (min(len(q1_words), len(q2_words)) - **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2 cwc_max = common_word_count / (max(len(q1_words), len(q2_words)) - **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2 csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops)) - **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2csc_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 Q2ctc_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 Q2ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))
- **last_word_eq** : Check if First word of both questions is equal or notlast_word_eq = int(q1_tokens[-1] == q2_tokens[-1])
- **first_word_eq** : Check if First word of both questions is equal or notfirst_word_eq = int(q1_tokens[0] == q2_tokens[0])
- **abs_len_diff** : Abs. length differenceabs_len_diff = abs(len(q1_tokens) - len(q2_tokens))
- **mean_len** : Average Token Length of both Questionsmean_len = (len(q1_tokens) + len(q2_tokens))/2
- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [0]:
```
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_DI
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DI
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DI
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DI
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE

    # 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
```

In [0]:
```
# 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)
```

In [0]:
```
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"

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

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

```
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"]
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["ques
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"],
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["questio
    return df
```

```
In [38]: if os.path.isfile('nlp_features_train_2.csv'):
           fe_2 = pd.read_csv("nlp_features_train_2.csv",encoding='latin-1')
           fe_2.fillna('')
         else:
           print("Extracting features for train:")
           fe_2 = pd.read_csv("df_fe_without_preprocessing_train_2.csv")
           fe_2 = extract_features(fe_2)
           fe_2.to_csv("nlp_features_train_2.csv", index=False)
           fe_2.head()

         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("train.csv")
             df = extract_features(df)
             df.to_csv("nlp_features_train.csv", index=False)
         df.head(2)
```

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

```
In [40]: fe_2.head()
```

Out[40]:

	id	qid1	qid2	...	fuzz_ratio	fuzz_partial_ratio	longest_substr_ratio
0	111330	182389	182390	...	57	59	0.166667
1	253083	42529	168758	...	54	60	0.269231
2	137682	138116	219411	...	46	54	0.176471
3	374163	505032	505033	...	23	50	0.296296
4	307595	431280	431281	...	46	65	0.454545

[5 rows x 32 columns]

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

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

```
In [41]: df_duplicate = fe_2[fe_2['is_duplicate'] == 1]
         dfp_nonduplicate = fe_2[fe_2['is_duplicate'] == 0]
```

```
# Converting 2d array of q1 and q2 and flatten the array: like {{1,2},{3,4}} to {1,2,3,
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) : 208968
Number of data points in class 0 (non duplicate pairs) : 357038
```

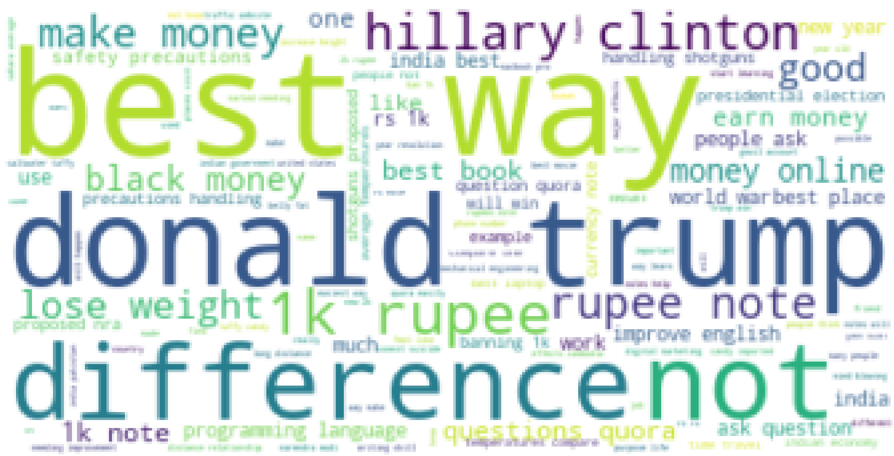
In [42]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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 : 11286191
Total number of words in non duplicate pair questions : 23227572
```

Word Clouds generated from duplicate pair question's text

In [43]:
```
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()
```

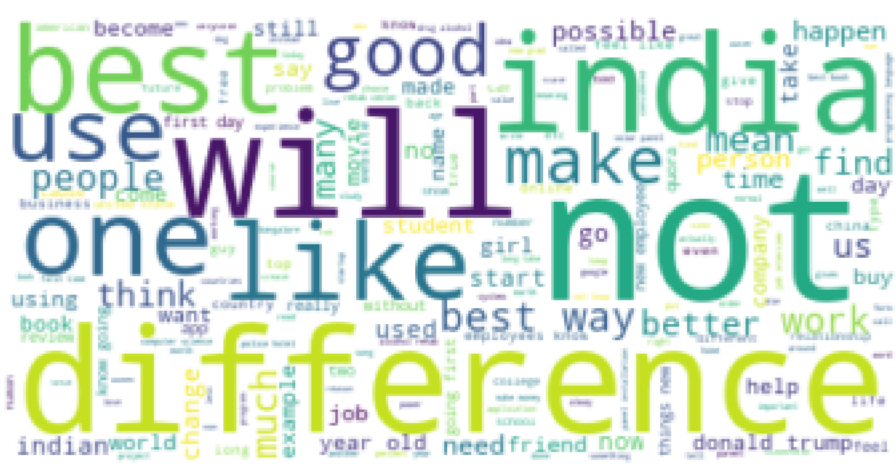
Word Cloud for Duplicate Question pairs



Word Clouds generated from non duplicate pair question's text

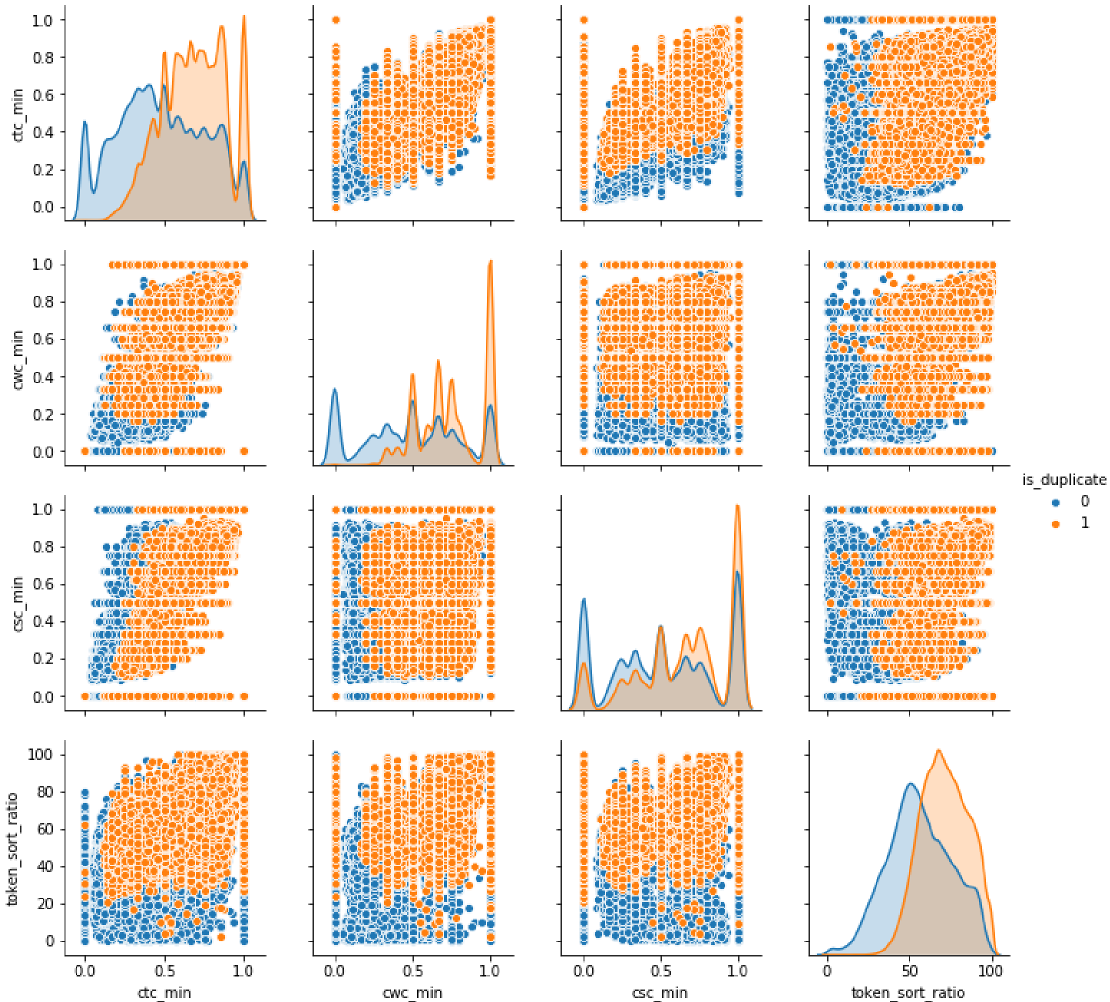
```
In [44]: wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
         # generate word cloud
         wc.generate(textn_w)
         print ("Word Cloud for non-Duplicate Question pairs:")
         plt.imshow(wc, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```

Word Cloud for non-Duplicate Question pairs:



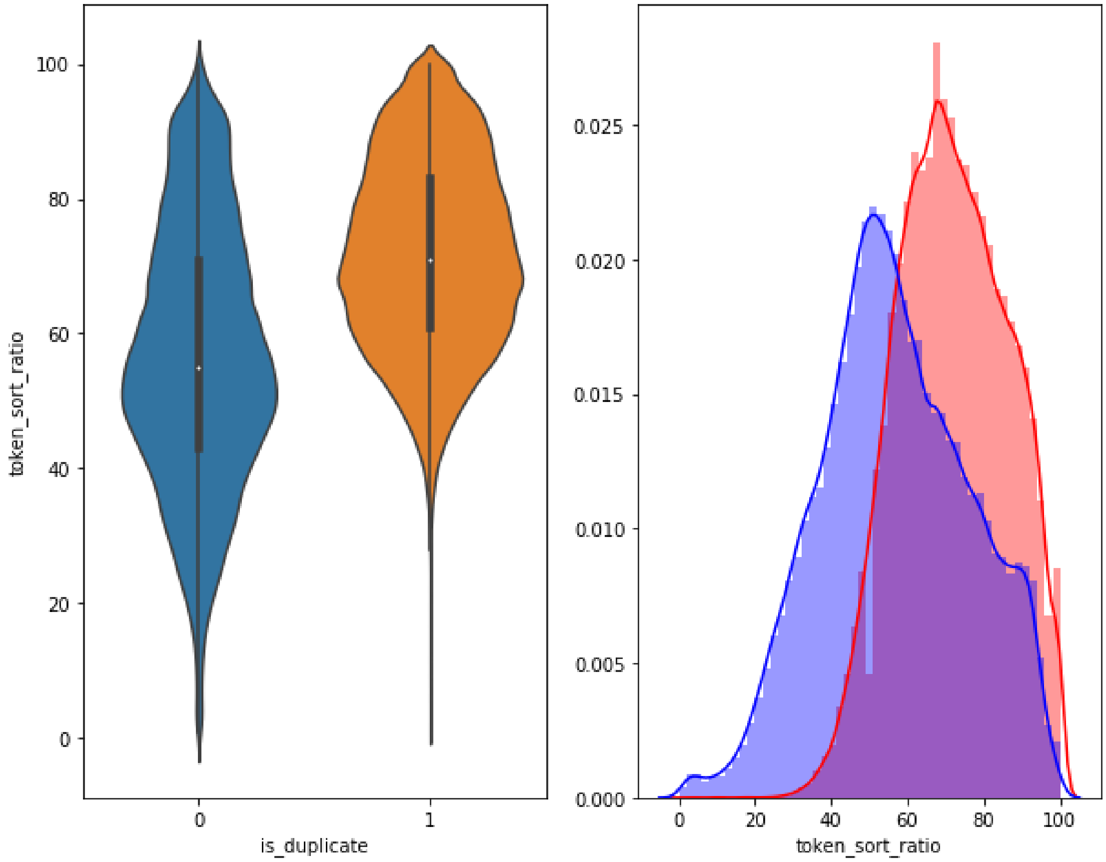
3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

```
In [45]: n = fe_2.shape[0]
         sns.pairplot(fe_2[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']
         plt.show()
```




```
In [47]: # 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 = fe_2[0:] , )

         plt.subplot(1,2,2)
         sns.distplot(fe_2[fe_2['is_duplicate'] == 1.0]['token_sort_ratio'][0:] , label = "1", c
```

```
sns.distplot(fe_2[fe_2['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" ,
plt.show()
```



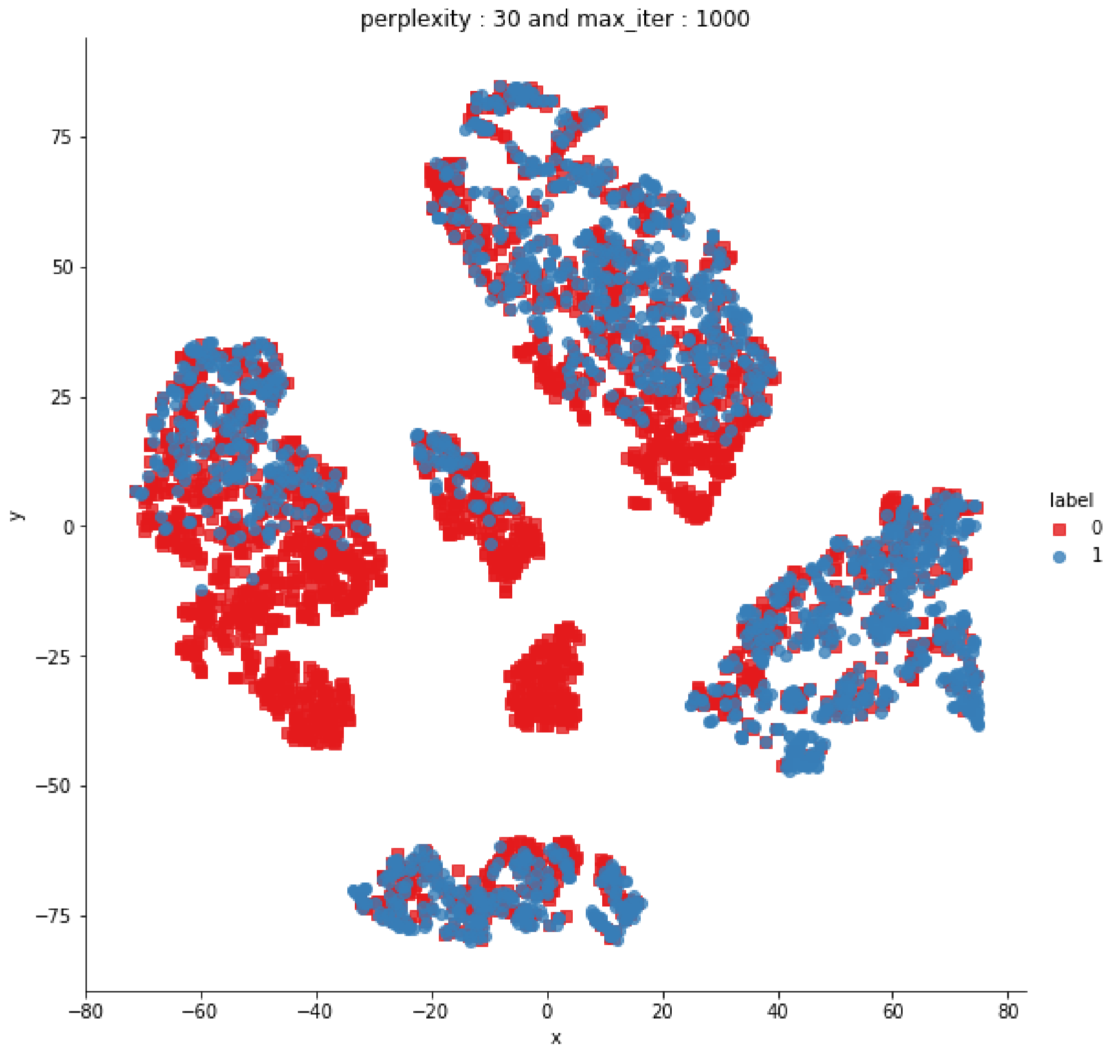
3.5.2 Visualization

```
In [0]: # Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the d

        from sklearn.preprocessing import MinMaxScaler

        dfp_subsampled = fe_2[0:5000]
        X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_m
        y = dfp_subsampled['is_duplicate'].values
```

```
In [51]: tsne2d = TSNE(
              n_components=2,
              init='random', # pca
              random_state=101,
              method='barnes_hut',
              n_iter=1000,
              verbose=2,
              angle=0.5
         ).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.021s...
[t-SNE] Computed neighbors for 5000 samples in 0.372s...
[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: 0.137823
[t-SNE] Computed conditional probabilities in 0.298s
[t-SNE] Iteration 50: error = 81.4102554, gradient norm = 0.0549030 (50 iterations in 2.294s)
[t-SNE] Iteration 100: error = 70.8214798, gradient norm = 0.0102546 (50 iterations in 1.733s)
[t-SNE] Iteration 150: error = 68.9858627, gradient norm = 0.0059349 (50 iterations in 1.646s)
[t-SNE] Iteration 200: error = 68.2404709, gradient norm = 0.0043851 (50 iterations in 1.670s)
[t-SNE] Iteration 250: error = 67.7898788, gradient norm = 0.0032174 (50 iterations in 1.716s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.789879
[t-SNE] Iteration 300: error = 1.8171654, gradient norm = 0.0011594 (50 iterations in 1.863s)
[t-SNE] Iteration 350: error = 1.4246323, gradient norm = 0.0004779 (50 iterations in 1.945s)
[t-SNE] Iteration 400: error = 1.2613554, gradient norm = 0.0002781 (50 iterations in 1.929s)
[t-SNE] Iteration 450: error = 1.1730725, gradient norm = 0.0001871 (50 iterations in 1.894s)
[t-SNE] Iteration 500: error = 1.1190690, gradient norm = 0.0001422 (50 iterations in 1.912s)
[t-SNE] Iteration 550: error = 1.0838553, gradient norm = 0.0001165 (50 iterations in 1.878s)
[t-SNE] Iteration 600: error = 1.0600814, gradient norm = 0.0001003 (50 iterations in 1.813s)
[t-SNE] Iteration 650: error = 1.0440030, gradient norm = 0.0000923 (50 iterations in 1.859s)
[t-SNE] Iteration 700: error = 1.0323466, gradient norm = 0.0000832 (50 iterations in 1.874s)
[t-SNE] Iteration 750: error = 1.0233592, gradient norm = 0.0000762 (50 iterations in 1.813s)
[t-SNE] Iteration 800: error = 1.0161228, gradient norm = 0.0000727 (50 iterations in 1.823s)
[t-SNE] Iteration 850: error = 1.0101161, gradient norm = 0.0000690 (50 iterations in 1.912s)
[t-SNE] Iteration 900: error = 1.0051531, gradient norm = 0.0000656 (50 iterations in 1.838s)
[t-SNE] Iteration 950: error = 1.0008084, gradient norm = 0.0000612 (50 iterations in 1.811s)
[t-SNE] Iteration 1000: error = 0.9970512, gradient norm = 0.0000571 (50 iterations in 1.820s)
[t-SNE] KL divergence after 1000 iterations: 0.997051
```

In [52]:
```python
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

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




```
In [53]: 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 0.015s...
[t-SNE] Computed neighbors for 5000 samples in 0.361s...
[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: 0.137823
[t-SNE] Computed conditional probabilities in 0.274s
[t-SNE] Iteration 50: error = 81.4588242, gradient norm = 0.0343189 (50 iterations in 12.080s)
[t-SNE] Iteration 100: error = 69.7078094, gradient norm = 0.0033843 (50 iterations in 5.569s)
[t-SNE] Iteration 150: error = 68.3959045, gradient norm = 0.0017439 (50 iterations in 4.759s)
[t-SNE] Iteration 200: error = 67.8454666, gradient norm = 0.0011496 (50 iterations in 4.837s)
[t-SNE] Iteration 250: error = 67.5082932, gradient norm = 0.0008905 (50 iterations in 4.875s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.508293
[t-SNE] Iteration 300: error = 1.5671912, gradient norm = 0.0007269 (50 iterations in 7.110s)
[t-SNE] Iteration 350: error = 1.2186174, gradient norm = 0.0002022 (50 iterations in 8.979s)
[t-SNE] Iteration 400: error = 1.0758561, gradient norm = 0.0001079 (50 iterations in 8.906s)
[t-SNE] Iteration 450: error = 1.0013239, gradient norm = 0.0000748 (50 iterations in 8.998s)
[t-SNE] Iteration 500: error = 0.9634528, gradient norm = 0.0000561 (50 iterations in 9.161s)
[t-SNE] Iteration 550: error = 0.9423503, gradient norm = 0.0000497 (50 iterations in 9.370s)
[t-SNE] Iteration 600: error = 0.9297930, gradient norm = 0.0000437 (50 iterations in 9.428s)
[t-SNE] Iteration 650: error = 0.9212062, gradient norm = 0.0000408 (50 iterations in 9.250s)
[t-SNE] Iteration 700: error = 0.9150742, gradient norm = 0.0000372 (50 iterations in 9.138s)
[t-SNE] Iteration 750: error = 0.9098119, gradient norm = 0.0000318 (50 iterations in 9.107s)
[t-SNE] Iteration 800: error = 0.9047760, gradient norm = 0.0000316 (50 iterations in 9.156s)
[t-SNE] Iteration 850: error = 0.9003149, gradient norm = 0.0000285 (50 iterations in 9.195s)
[t-SNE] Iteration 900: error = 0.8962525, gradient norm = 0.0000269 (50 iterations in 9.243s)
[t-SNE] Iteration 950: error = 0.8921583, gradient norm = 0.0000273 (50 iterations in 9.281s)
[t-SNE] Iteration 1000: error = 0.8889708, gradient norm = 0.0000254 (50 iterations in 9.317s)
[t-SNE] KL divergence after 1000 iterations: 0.888971
```

In [54]:
```python
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')
```

```
In [0]: from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer
        from tqdm import tqdm

        import spacy
```

```
In [0]: fe_2['question1'] = fe_2['question1'].apply(lambda x: str(x))
        fe_2['question2'] = fe_2['question2'].apply(lambda x: str(x))
```

```
In [57]: questions_train = list(fe_2['question1']) + list(fe_2['question2'])

         vectorizer_tfidf_questions = TfidfVectorizer(min_df=40)
         vectorizer_tfidf_questions.fit(questions_train)
```

```
Out[57]: TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                         dtype=<class 'numpy.float64'>, encoding='utf-8',
                         input='content', lowercase=True, max_df=1.0, max_features=None,
                         min_df=40, ngram_range=(1, 1), norm='l2', preprocessor=None,
                         smooth_idf=True, stop_words=None, strip_accents=None,
                         sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                         tokenizer=None, use_idf=True, vocabulary=None)
```

```
In [59]: questions1_tfidf_train = vectorizer_tfidf_questions.transform(fe_2['question1'])
         print("Shape of matrix after one hot encoding ",questions1_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (283003, 7762)
```

```
In [60]: questions2_tfidf_train = vectorizer_tfidf_questions.transform(fe_2['question2'])
         print("Shape of matrix after one hot encoding ",questions2_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (283003, 7762)
```

0.2 Final Trainset

```
In [64]: fe_2.drop(['id','is_duplicate','qid1','qid2','question1','question2'], axis=1, inplace=
         fe_2.shape
```

```
Out[64]: (283003, 26)
```

```
In [65]: from scipy.sparse import hstack
         X_tr = hstack((fe_2,questions1_tfidf_train,questions2_tfidf_train)).tocsr()
         print(X_tr.shape, y_train.shape)
         print("="*100)
```

```
(283003, 15550) (283003,)
==============================================================================================
```

0.3 Final TestSet

```
In [68]: if os.path.isfile('df_fe_without_preprocessing_test_2.csv'):
           X_test = pd.read_csv("df_fe_without_preprocessing_test_2.csv",encoding='latin-1')
         else:
           X_test['freq_qid1'] = X_test.groupby('qid1')['qid1'].transform('count')
           X_test['freq_qid2'] = X_test.groupby('qid2')['qid2'].transform('count')
           X_test['q1len'] = X_test['question1'].str.len()
           X_test['q2len'] = X_test['question2'].str.len()
           X_test['q1_n_words'] = X_test['question1'].apply(lambda row: len(row.split(" ")))
           X_test['q2_n_words'] = X_test['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)
           X_test['word_Common'] = X_test.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))
           X_test['word_Total'] = X_test.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))

           X_test['word_share'] = X_test.apply(normalized_word_share, axis=1)
           X_test['freq_q1+q2'] = X_test['freq_qid1']+X_test['freq_qid2']
           X_test['freq_q1-q2'] = abs(X_test['freq_qid1']-X_test['freq_qid2'])
           X_test.to_csv("df_fe_without_preprocessing_test_2.csv", index=False)
         X_test.head()
```

```
Out[68]:             id    qid1    qid2  ...  word_share  freq_q1+q2  freq_q1-q2
         283650  283650   64942    5508  ...    0.200000           3           1
         369047  369047  499447  117115  ...    0.277778           3           1
         98525    98525  163710  163711  ...    0.277778           2           0
         196363  196363  251287  297048  ...    0.400000           4           2
         35950    35950   65614   65615  ...    0.475000           3           1

         [5 rows x 17 columns]
```

```
In [69]: if os.path.isfile('nlp_features_test_2.csv'):
           fe_3 = pd.read_csv("nlp_features_test_2.csv",encoding='latin-1')
           fe_3.fillna('')
```

```
         else:
           print("Extracting features for test:")
           fe_3 = pd.read_csv("df_fe_without_preprocessing_test_2.csv")
           fe_3 = extract_features(fe_3)
           fe_3.to_csv("nlp_features_test_2.csv", index=False)
           fe_3.head()
```

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

```
In [0]: fe_3['question1'] = fe_3['question1'].apply(lambda x: str(x))
        fe_3['question2'] = fe_3['question2'].apply(lambda x: str(x))
```

```
In [71]: questions1_tfidf_test = vectorizer_tfidf_questions.transform(fe_3['question1'])
         print("Shape of matrix after one hot encoding ",questions1_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (121287, 7762)
```

```
In [72]: questions2_tfidf_test = vectorizer_tfidf_questions.transform(fe_3['question2'])
         print("Shape of matrix after one hot encoding ",questions2_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (121287, 7762)
```

```
In [0]: fe_3.drop(['id','is_duplicate','qid1','qid2','question1','question2'], axis=1, inplace=T
```

```
In [74]: X_te = hstack((fe_3,questions1_tfidf_test,questions2_tfidf_test)).tocsr()
         print(X_te.shape, y_test.shape)
         print("="*100)
```

```
(121287, 15550) (121287,)
================================================================================================
```

```
In [0]: # This function plots the confusion matrices given y_i, y_i_hat.
        from sklearn.metrics import confusion_matrix

        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 predict

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

            B =(C/C.sum(axis=0))
            #divid each element of the confusion matrix with the sum of elements in that row
```

```
plt.figure(figsize=(20,4))

labels = [1,2]
# representing A in heatmap format
cmap=sns.light_palette("blue")
plt.subplot(1, 3, 1)
sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=lab
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=lab
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=lab
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)

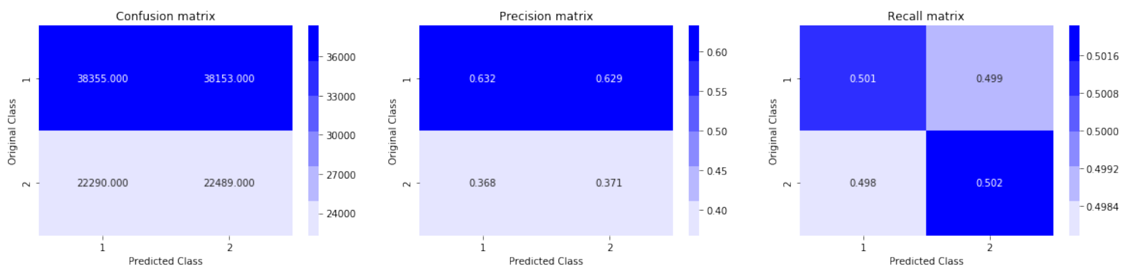
```
In [78]: from collections import Counter

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

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in test data ----------
Class 0:  0.6308013224830361 Class 1:  0.3691986775169639
```

```
In [82]: # 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
```

```
from sklearn.metrics.classification import accuracy_score, log_loss

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-1

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

```
Log loss on Test Data using Random Model 0.8846687907834682
```

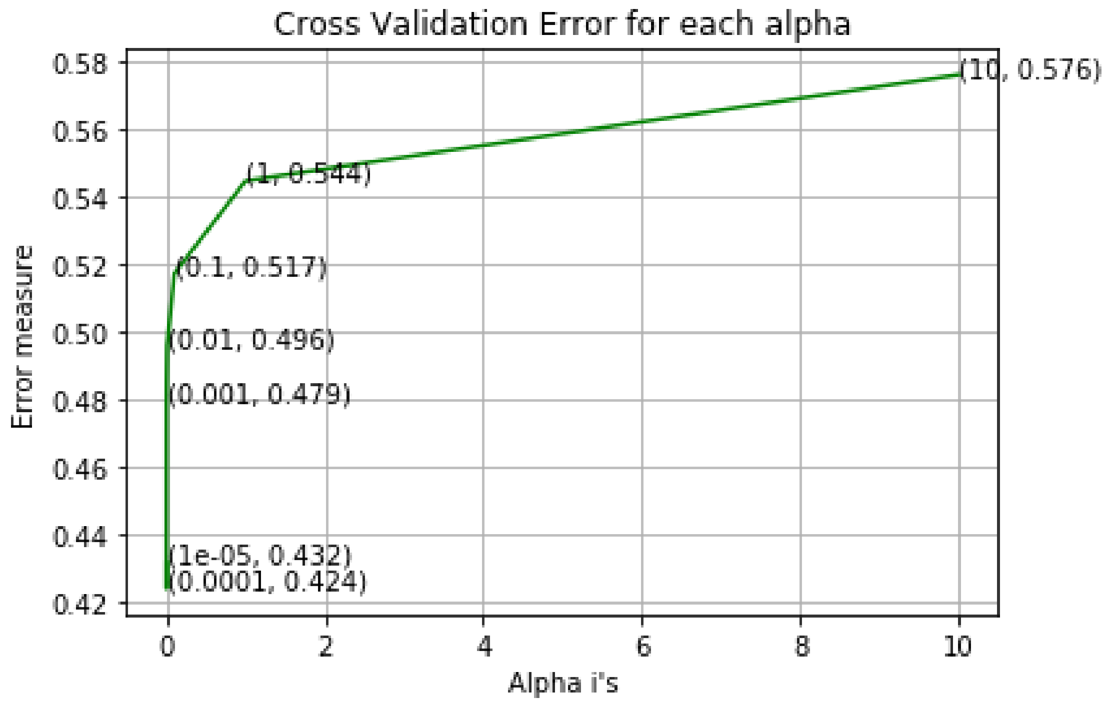


0.4 Logistic Regression with hyperparameter tuning

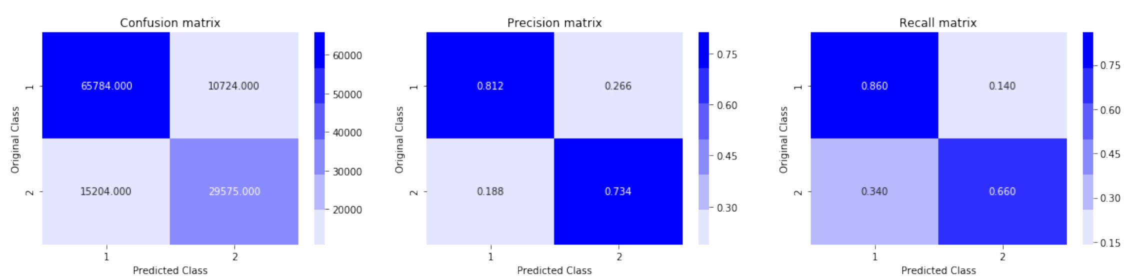
```
In [88]: from sklearn.model_selection import cross_val_score
         from sklearn.linear_model import SGDClassifier
         from mlxtend.classifier import StackingClassifier

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

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

         log_error_array=[]
         for i in alpha:
             clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf.fit(X_tr, y_train)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(X_tr, y_train)
             predict_y = sig_clf.predict_proba(X_te)
             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, l

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

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

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10]
For values of alpha =  1e-05 The log loss is: 0.4315401684692288
For values of alpha =  0.0001 The log loss is: 0.42386147721888673
For values of alpha =  0.001 The log loss is: 0.4792501650759294
For values of alpha =  0.01 The log loss is: 0.4956402799397207
For values of alpha =  0.1 The log loss is: 0.5170289534134559
For values of alpha =  1 The log loss is: 0.5444887618742672
For values of alpha =  10 The log loss is: 0.5758380952275102
```




```
For values of best alpha =  0.0001 The train log loss is: 0.3891075754840594
For values of best alpha =  0.0001 The test log loss is: 0.42386147721888673
Total number of data points : 121287
```

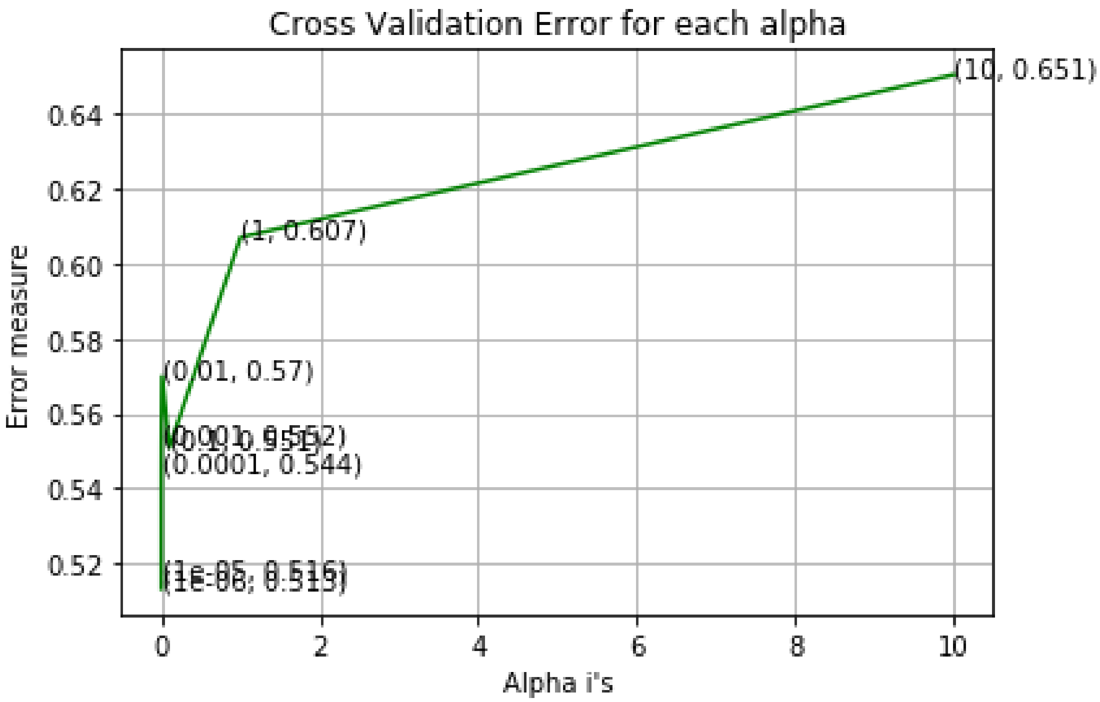


4.6 Linear SVM with hyperparameter tuning

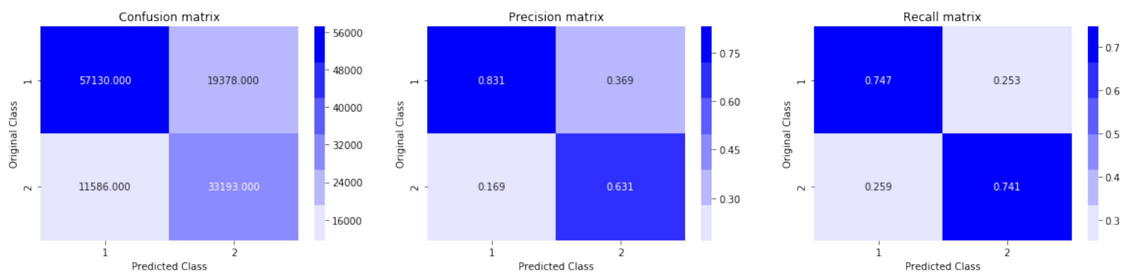
```
In [90]: alpha = [10 ** x for x in range(-6, 2)]

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

```
        sig_clf.fit(X_tr, y_train)
        predict_y = sig_clf.predict_proba(X_te)
        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, l

    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=4
    clf.fit(X_tr, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr, y_train)

    predict_y = sig_clf.predict_proba(X_tr)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
    predict_y = sig_clf.predict_proba(X_te)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
    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-06 The log loss is: 0.513211839689834
For values of alpha =  1e-05 The log loss is: 0.5160174069478672
For values of alpha =  0.0001 The log loss is: 0.5444625109383455
For values of alpha =  0.001 The log loss is: 0.5523981514632448
For values of alpha =  0.01 The log loss is: 0.5699434878358832
For values of alpha =  0.1 The log loss is: 0.5508106227904552
For values of alpha =  1 The log loss is: 0.6071354556244608
For values of alpha =  10 The log loss is: 0.6505381905588339
```




```
For values of best alpha =  1e-06 The train log loss is: 0.4362632953008896
For values of best alpha =  1e-06 The test log loss is: 0.513211839689834
Total number of data points : 121287
```



4.7 XGBoost

```
In [0]: os.environ['KMP_DUPLICATE_LIB_OK']='True'
        final_x = X_tr[0:70000]
        final_y = y_train[0:70000]

        final_x_te = X_te[0:30000]
        final_y_te = y_test[0:30000]
```

```
In [0]: params = {
            'max_depth': [3, 4, 5, 6, 7, 8],
            'eta' : [0.01, 0.02, 0.05, 0.1, 0.2, 0.3],
            'n_estimators' : [100, 200, 300, 400, 500],
            'gamma': [0, 0.5, 1, 1.5, 2, 5]
        }
```

```
In [0]: import xgboost as xgb
        xgb = XGBClassifier(nthread=1)
```

```
In [115]: from sklearn.model_selection import StratifiedKFold
          from sklearn.model_selection import RandomizedSearchCV

          folds = 3
          param_comb = 4
          skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 42)
          random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=param_comb,

          random_search.fit(final_x, final_y)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed: 13.1min finished
```

```
Out[115]: RandomizedSearchCV(cv=<generator object _BaseKFold.split at 0x7f3ba43b3468>,
                             error_score='raise-deprecating',
                             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                                     colsample_bylevel=1,
                                                     colsample_bynode=1,
                                                     colsample_bytree=1, gamma=0,
                                                     learning_rate=0.1, max_delta_step=0,
                                                     max_depth=3, min_child_weight=1,
                                                     missing=None, n_estimators=100,
                                                     n_jobs=1, nthread=1,
                                                     objective...
                                                     reg_lambda=1, scale_pos_weight=1,
                                                     seed=None, silent=None, subsample=1,
                                                     verbosity=1),
                             iid='warn', n_iter=4, n_jobs=-1,
                             param_distributions={'eta': [0.01, 0.02, 0.05, 0.1, 0.2,
                                                          0.3],
                                                  'gamma': [0, 0.5, 1, 1.5, 2, 5],
                                                  'max_depth': [3, 4, 5, 6, 7, 8],
                                                  'n_estimators': [100, 200, 300, 400,
                                                                   500]},
                             pre_dispatch='2*n_jobs', random_state=42, refit=True,
                             return_train_score=False, scoring='neg_log_loss', verbose=2)
```

In [116]:
```
print('\n Best estimator:')
print(random_search.best_estimator_)
print('\n Best hyperparameters:')
print(random_search.best_params_)
```

```
 Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.3, gamma=2,
              learning_rate=0.1, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=1,
              nthread=1, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)

 Best hyperparameters:
{'n_estimators': 500, 'max_depth': 7, 'gamma': 2, 'eta': 0.3}
```

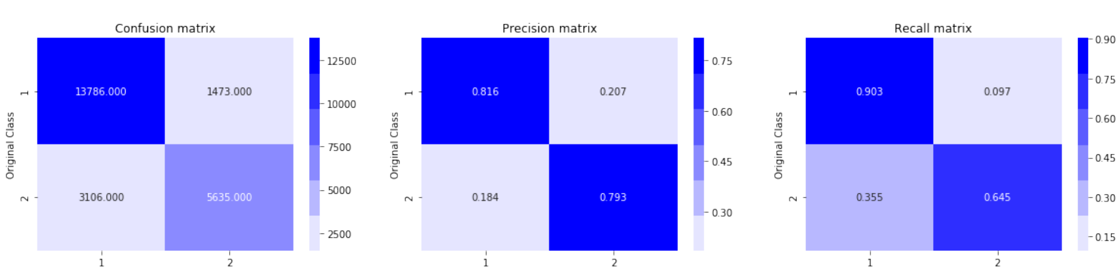
In [117]:
```
import xgboost as xgb

params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.3
params['max_depth'] = 7
params['n_estimators'] = 500
params['gamma'] = 2


d_train = xgb.DMatrix(final_x, label=final_y)
d_test = xgb.DMatrix(final_x_te, label=final_y_te)

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

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

xgdmat = xgb.DMatrix(final_x,final_y)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(final_y_te, predict_y, labels=clf.classes_, eps
```

```
[0]	train-logloss:0.571543	valid-logloss:0.585792
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]	train-logloss:0.35614	valid-logloss:0.409056
[20]	train-logloss:0.334579	valid-logloss:0.395619
[30]	train-logloss:0.32338	valid-logloss:0.39044
```

```
[40]        train-logloss:0.312399       valid-logloss:0.386012
[50]        train-logloss:0.305032       valid-logloss:0.384032
[60]        train-logloss:0.299893       valid-logloss:0.383381
[70]        train-logloss:0.293106       valid-logloss:0.381638
[80]        train-logloss:0.288059       valid-logloss:0.380058
[90]        train-logloss:0.283002       valid-logloss:0.379624
[100]        train-logloss:0.279113       valid-logloss:0.379045
[110]        train-logloss:0.274863       valid-logloss:0.377947
[120]        train-logloss:0.271661       valid-logloss:0.377594
[130]        train-logloss:0.268436       valid-logloss:0.37728
[140]        train-logloss:0.265267       valid-logloss:0.377006
[150]        train-logloss:0.262871       valid-logloss:0.376524
[160]        train-logloss:0.25842       valid-logloss:0.375931
[170]        train-logloss:0.25578       valid-logloss:0.375486
[180]        train-logloss:0.253534       valid-logloss:0.375155
[190]        train-logloss:0.251223       valid-logloss:0.374837
[200]        train-logloss:0.248146       valid-logloss:0.374503
[210]        train-logloss:0.24622       valid-logloss:0.374427
[220]        train-logloss:0.244236       valid-logloss:0.374315
[230]        train-logloss:0.242632       valid-logloss:0.374199
[240]        train-logloss:0.240581       valid-logloss:0.373904
[250]        train-logloss:0.237653       valid-logloss:0.373735
[260]        train-logloss:0.235944       valid-logloss:0.373649
[270]        train-logloss:0.232724       valid-logloss:0.372947
[280]        train-logloss:0.229769       valid-logloss:0.372802
[290]        train-logloss:0.227474       valid-logloss:0.372903
Stopping. Best iteration:
[278]        train-logloss:0.230807       valid-logloss:0.372595

The test log loss is: 0.37288770671097543
```

In [110]:
```python
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(final_y_te, predicted_y)
```

```
Total number of data points : 24000
```



```
In [111]: from prettytable import PrettyTable
          #If you get a ModuleNotFoundError error , install prettytable using: pip3 install pret

          x = PrettyTable()
          x.field_names = ["Vectorizer", "Model", "Train log loss", "Test log loss"]
          x.add_row(["TFIDF", "Random Model", "-----", 0.8846])
          x.add_row(["TFIDF", "Logistic Regression", 0.38, 0.4230])
          x.add_row(["TFIDF", "Linear SVM", 0.434, 0.5103])
          x.add_row(["TFIDF", "XGBoost", 0.258, 0.372])

          print(x)
```

```
+------------+---------------------+----------------+---------------+
| Vectorizer |        Model        | Train log loss | Test log loss |
+------------+---------------------+----------------+---------------+
|   TFIDF    |     Random Model    |     -----      |     0.8846    |
|   TFIDF    | Logistic Regression |      0.38      |     0.423     |
|   TFIDF    |      Linear SVM     |     0.434      |     0.5103    |
|   TFIDF    |       XGBoost       |     0.258      |     0.378     |
+------------+---------------------+----------------+---------------+
```

0.5 Step by Step Procedure

- Understanding the Problem Statement

- Checking the dataset
- Load the given dataset.
- It contains 5 columns with question id1, id2, question1,question2 and is_duplicate features.
- Number of rows 404,290

- Exploratory Data Analysis
- Distribution of data points among output classes - Question pairs which are not Similar (is_duplicate = 0) is 63.08%. Question pairs which are Similar(is_duplicate = 1) 36.92%
- Number of unique questions that appear more than one time: 111780
- Checked for NULL values and deleted the rows containing them as there are only 3.

- Splitting Train and test data to avoid data leakage before feature engineering.
- Shape of train data (283003, 6)
- Shape of test data (121287, 6)

- Basic Feature Extraction/Engineering
- ____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

- Cleaning the dataset
- Remove stopwords
- Remove punctions
- Stem the tokens(Potter Stemmer)
- Remove HTML tags
- Expand Contractions

- Advance Feature Engineering

- **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 Q2csc_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 Q2ctc_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 Q2ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

- **last_word_eq** : Check if First word of both questions is equal or notlast_word_eq = int(q1_tokens[-1] == q2_tokens[-1])

- **first_word_eq** : Check if First word of both questions is equal or notfirst_word_eq = int(q1_tokens[0] == q2_tokens[0])

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

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

- **fuzz_ratio**

- **fuzz_partial_ratio**

- **token_sort_ratio**

- **token_set_ratio**

- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2

- Vectorization using TFIDF
- Resulting in nearly 7.7k features
- Prepare the Train and Test Dataset

- Build a Random Model to estimate the Maximum Loss a Model can have
- Plot the Confusion matrix, Precision matrix, Recall matrix
- Got 0.88 as the upper bound for Log- Loss

- Build a Logistic Regression Model with Hyperparameter Tuning
- Set a range of values for alpha ranging from 10^-5 to 10^2
- Pick the best performing parameter on Train Data based on Loss on Train-CV.
- Consider Log loss and L2 penalty.
- Calculate the Loss with the obtained parameter on Test Data.
- Plot the Confusion matrix, Precision matrix, Recall matrix.

- Build a Linear SVM Model with Hyperparameter Tuning
- Set a range of values for alpha ranging from 10^-6 to 10^2
- Pick the best performing parameter on Train Data based on Loss on Train-CV.
- Consider Hinge Loss and L1 penalty.
- Calculate the Loss with the obtained parameter on Test Data.
- Plot the Confusion matrix, Precision matrix, Recall matrix.

- Build a XGBoost Model with Hyperparameter Tuning
- Assume 70,000 data points in train and 30K points in test data to avoid longer durations of wait for computations.
- Set a range of values for various parameters.
- Apply Randomized Search CV as it takes lesser time to compute than Grid Search CV.
- Use 3 fold Cross Vaidation.
- Pick the best performing parameter on Train Data based on Loss on Train-CV.
- Calculate the Loss with the obtained parameter on Test Data.
- Plot the Confusion matrix, Precision matrix, Recall matrix

In [0]: