Document Created date:08-08-2019

Quora

Question Pair Similarity prediction

Machine Learning Application

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.

> Credits: Kaggle

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.

1.2 Sources/Useful Links

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

Useful Links

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments (https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments)
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0 (https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0)
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning (https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning)
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30 (https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30)

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 of choice.
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 indi
a?","What is the step by step guide to invest in share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamond?","What would h
appen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond bac
k?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a great
geologist?","1"
```

```
"11","23","24","How do I read and find my YouTube comments?","How can I see all
my Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

```
I followed below steps
```

- Understanding the Businessreal world problem
- Understanding Real-world/Business objectives and constraints.
- Understanding Data overview
- Mapping the real-world problem to an ML problem
- Defining Performance Metric as per the ML problem
- importing required libraries
- Reading the data (train.csv)
- Exploratory Data Analysis
  - Distribution of data points among output classes
  - Number of unique questions
  - checking for duplicates
  - Number of occurrences of each question
  - checking for null values
- Basic Feature Extraction (before cleaning)
  - Analysis of some of the extracted features
  - Ploting Feature: word_share (violin Plots)
  - Ploting Feature: word_common(violin Plots
- Preprocessing of Text:
  - Removing html tags

    - Removing Punctuations
    - Performing stemming
    - Removing Stopwords
    - Expanding contractions etc.
- Advanced Feature Extraction (NLP and Fuzzy Features)
- Analysis of extracted features
    - Ploting wordclouds
    - Word Clouds generated from duplicate pair question's text
    - Word Clouds generated from non duplicate pair question's text
- Ploting Pair plots and violin polts of features ('ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio')
- Apply TSNE method to reduce the Dimentionality for 15 Features and ploting
- Featurizing text data with tfidf weighted word-vectors
- Saving final features to csv file
- Reading saved csv data from file and storing into SQL Table
- Spliting data into train, test, cv\
- Extraction features from train and test data frame
- defining Confusion Matrix , Precision matrix, Recall matrix
- Building a random model (Finding worst-case log-loss)
    - Random Model - Hyper parameter Tuning - Ploting with Loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Apply Linear SVM model
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Apply XGBoost on Random model
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Reading Data from file (nlp features train.csv)
- Data Cleaning - droping coloumns and merding data frames , filling the nan values
- Splitting the data into train, test, cv
- TFIDF vectorizer on Questions Text Dat
- TFIDF Vectorizer on Train data _ question1 and question2
- Combining our tfidf and features into one using hstack from scipy
- Defining confusion matrix , precision matrix, recall precision
- Building Random model (worse case log loss) - TFID
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Apply Logistic Regression model
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Apply Liner SVM model
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Apply XGBoost model
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- TFIDF W2V 50 K Datapoints loading
    - Hyper parameter Tuning - ploting with loss
    - ploting Confusion matrix , Precision matrix, Recall matrix
- Observation on overall model performences (Conclusion)
- Ploting the performences by table format.

---

3. Exploratory Data Analysis

In [1]:

```
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
```

In [70]:

```
pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7
eaa1beab2b9323073815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-n
one-any.whl (https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7e
aa1beab2b9323073815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-no
ne-any.whl)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
Note: you may need to restart the kernel to use updated packages.
```

In [77]:

```
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
import nltk
nltk.download('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
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\battu1989.WINDOWS-
[nltk_data]     BATTU19\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

3.1 Reading data and basic stats

In [52]:

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

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

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

In [150]:

```
df.head()
```

Out[150]:

	id	qid1	qid2	question1	question2	is_duplicate	q1_feats_m	q2_feat
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	[-6.179506778717041, 37.45073118805885, -67.92...	[-14.616980731487 59.75548753142 -5
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	[9.236667931079865, -80.37141644954681, -45.78...	[-3.5657422859221 -16.844570636749
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	[97.54683184623718, 22.97219370305538, -39.558...	[156.8336295336 59.99189615249 -8.41
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24} [/math] i...	0	[57.58697843551636, -22.017089188098907, -4.59...	[41.47243919968 56.71731689572 31.53
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	[83.1857842206955, -40.50698482990265, -83.403...	[-14.446974992752 -4.33825546503 -7

In [151]:

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 8 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404290 non-null object
question2       404290 non-null object
is_duplicate    404290 non-null int64
q1_feats_m      404290 non-null object
q2_feats_m      404290 non-null object
dtypes: int64(4), object(4)
memory usage: 24.7+ MB
None
```

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

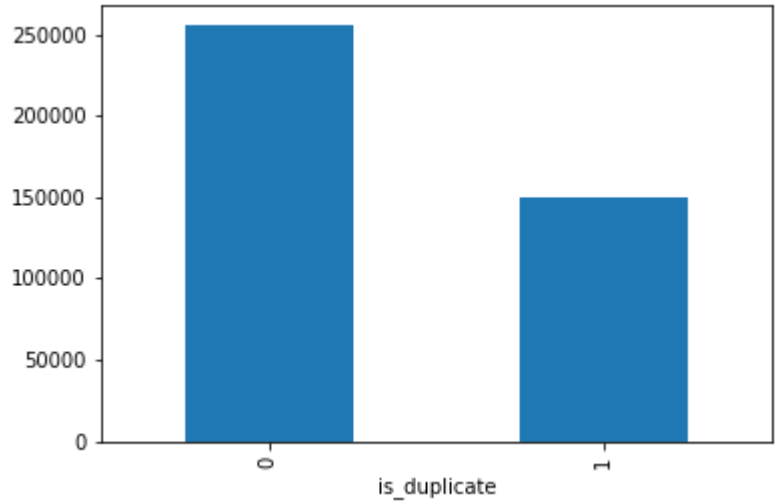
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [55]:

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

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e9f00bbb08>
```



In [56]:

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

```
~> Total number of question pairs for training:
   404290
```

In [57]:

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

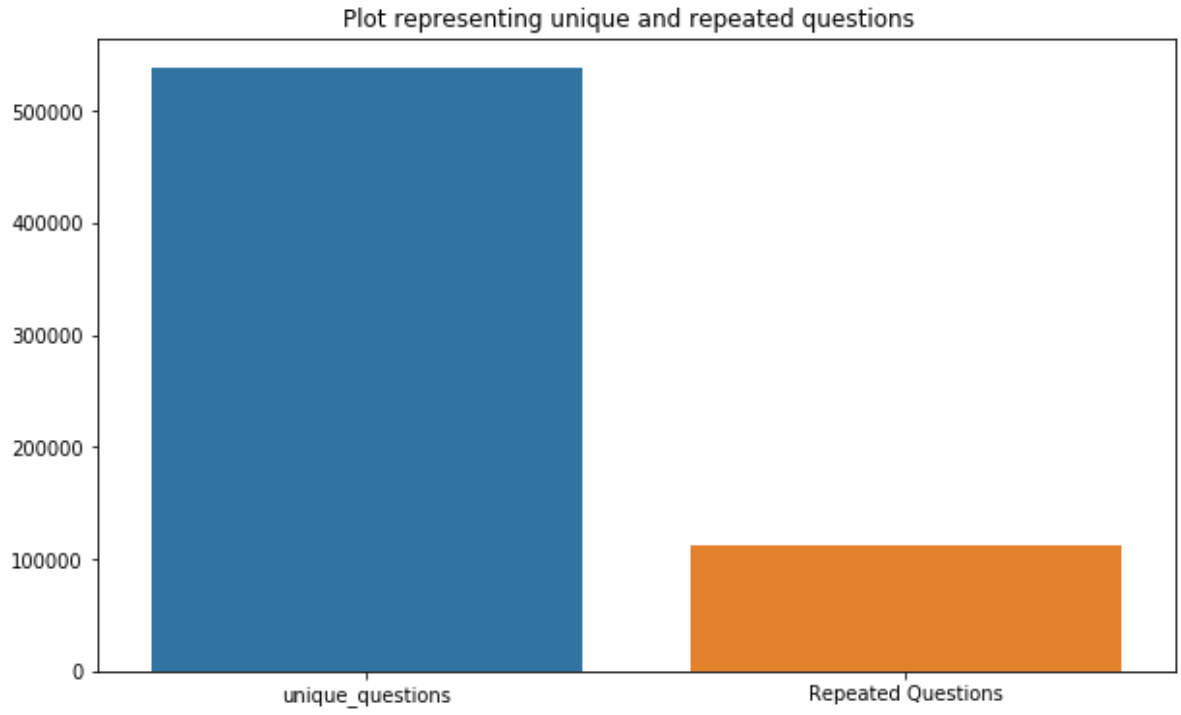
In [58]:

```
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 (20.7795
3945937505%)

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

In [59]:

```
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 [60]:

```
#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])
```

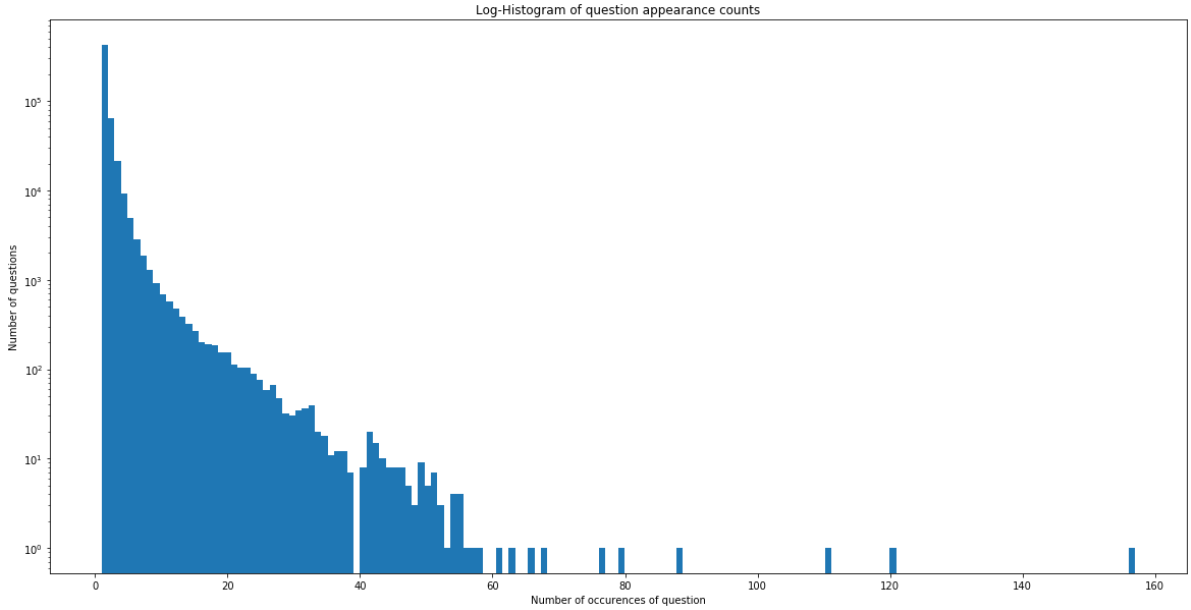
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [61]:

```
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 [62]:

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

```
            id    qid1    qid2                          question1  \
105780  105780  174363  174364    How can I develop android app?
201841  201841  303951  174364  How can I create an Android app?
363362  363362  493340  493341                               NaN

                                                 question2  is_duplicate
105780                                                 NaN             0
201841                                                 NaN             0
363362  My Chinese name is Haichao Yu. What English na...              0
```

- There are two rows with null values in question2

In [63]:

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

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

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

In [64]:

```
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()
```

Out[64]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n
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	50	65	
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [65]:

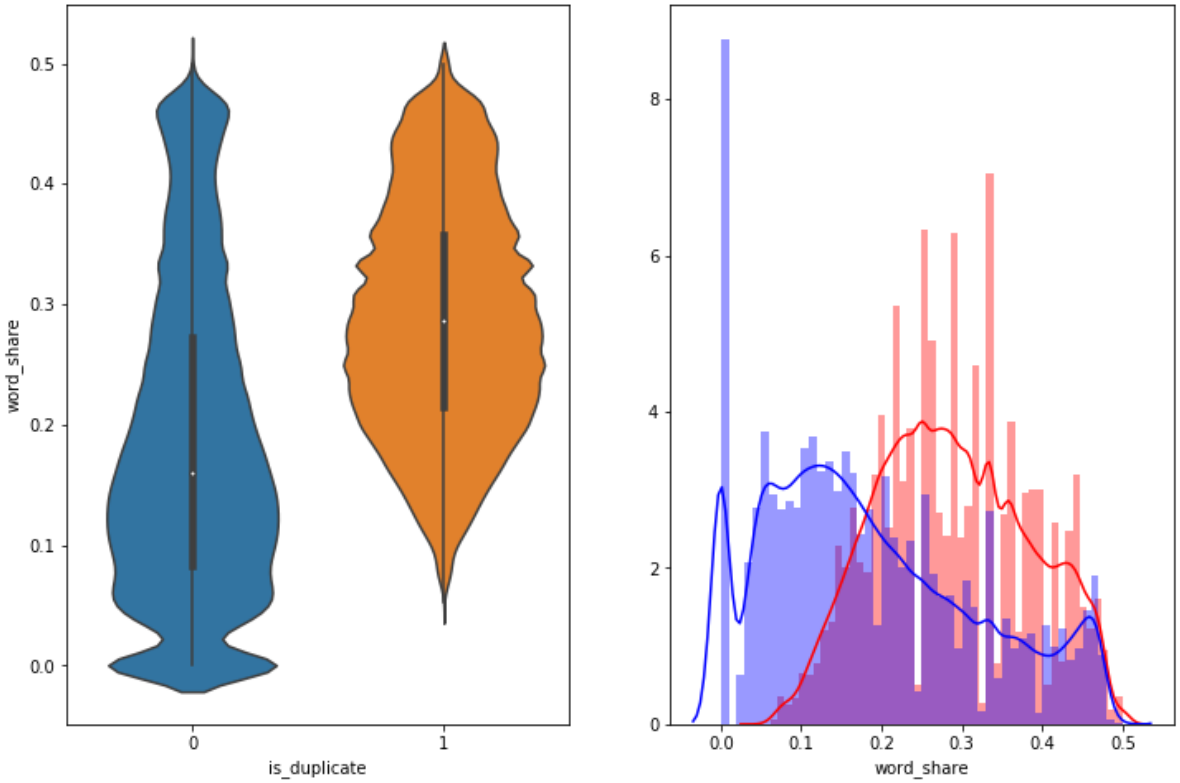
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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

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

```
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
```

3.3.1.1 Feature: word_share

In [66]:

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

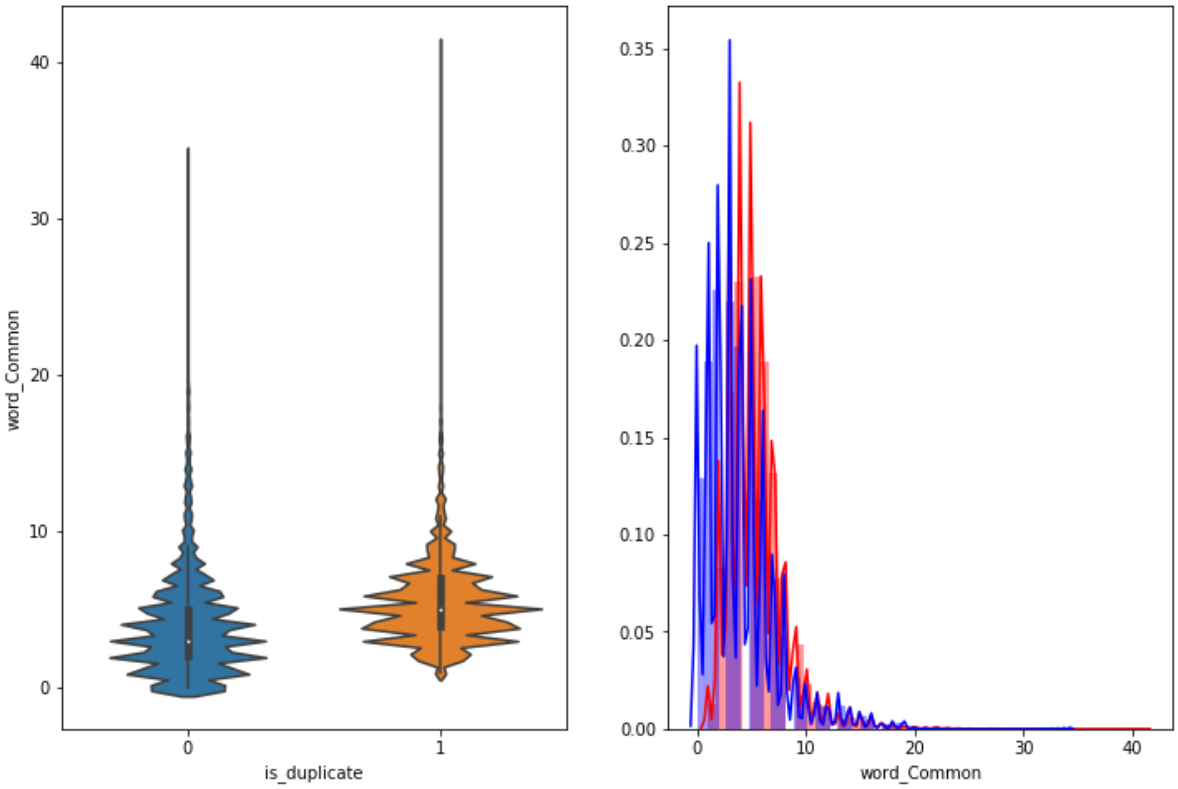


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [67]:

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



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

3.4 Preprocessing of Text

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

In [76]:

```
# 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 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
```

- Function to Compute and get the features : With 2 parameters of Question 1 and Question 2

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 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 (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (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 Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [78]:

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

    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

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

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

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

# get the Longest Common sub string

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

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

```
    print("token features...")

    # Merging Features with dataset

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

    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-c
    # 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 t
    # 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["que
    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["questi
    return df
```

In [79]:

```
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)
```

Out[79]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988

2 rows × 21 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 [81]:

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

# Converting 2d array of q1 and q2 and flatten the array: like {{1,2},{3,4}} to {1,2,3,
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
# cite :https://stackoverflow.com/questions/27092833/unicodeencodeerror-charmap-codec-c
np.savetxt('train_p.txt', p, delimiter=' ', encoding="utf-8", fmt='%s')
np.savetxt('train_n.txt', n, delimiter=' ', encoding="utf-8", fmt='%s')
```

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

In [82]:

```
# 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))
```

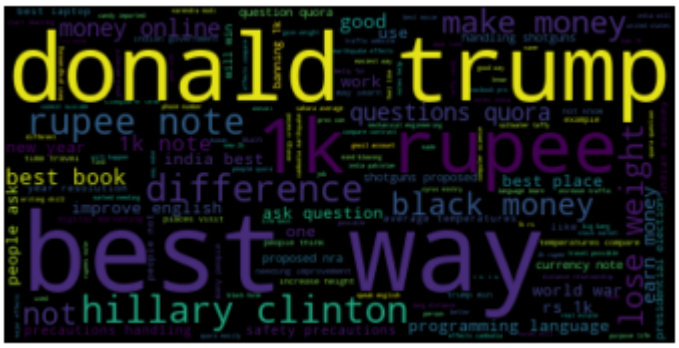
```
Total number of words in duplicate pair questions : 16110303
Total number of words in non duplicate pair questions : 33194892
```

Word Clouds generated from duplicate pair question's text

In [83]:

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

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



Word Clouds generated from non duplicate pair question's text

In [84]:

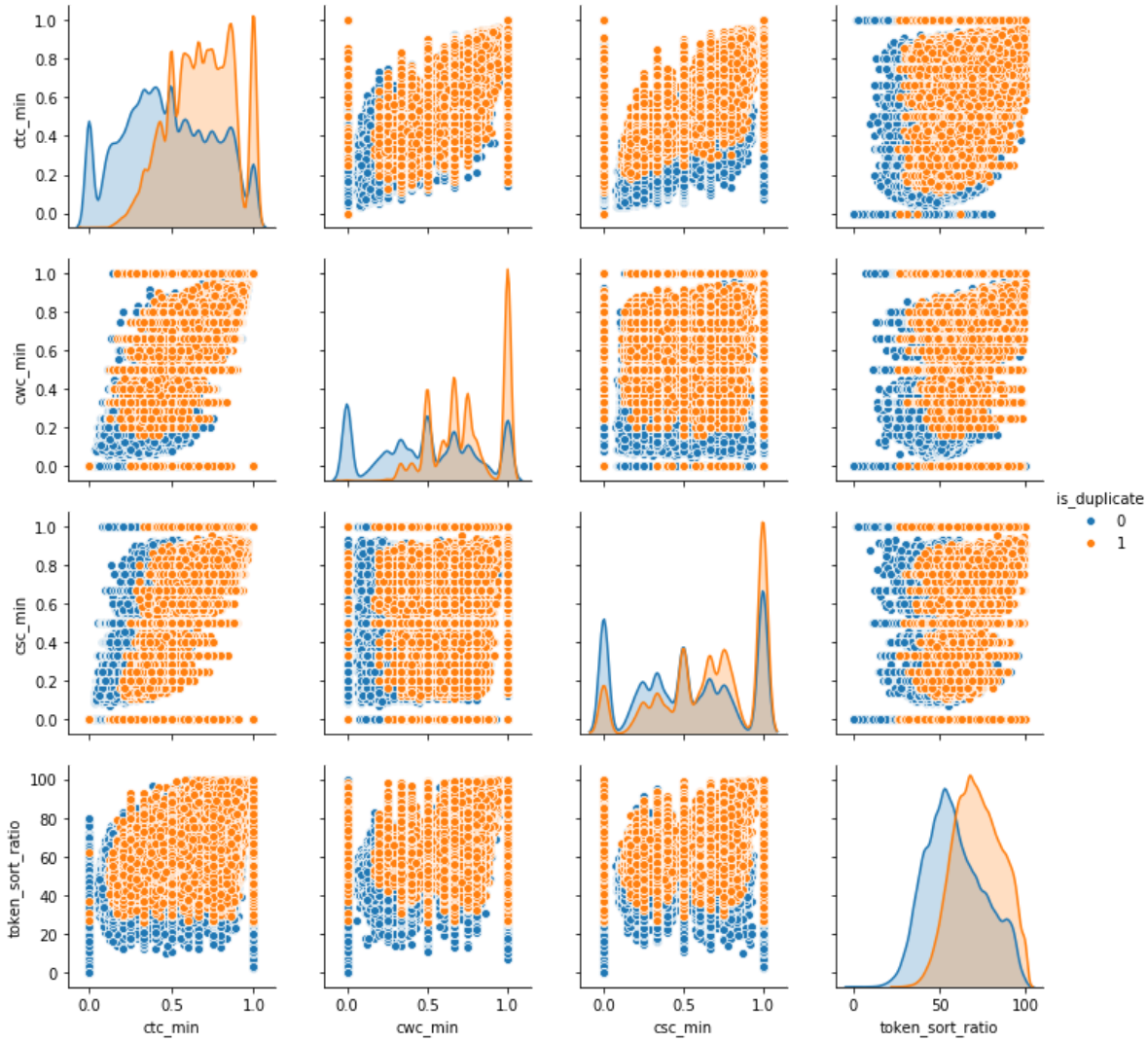
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

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

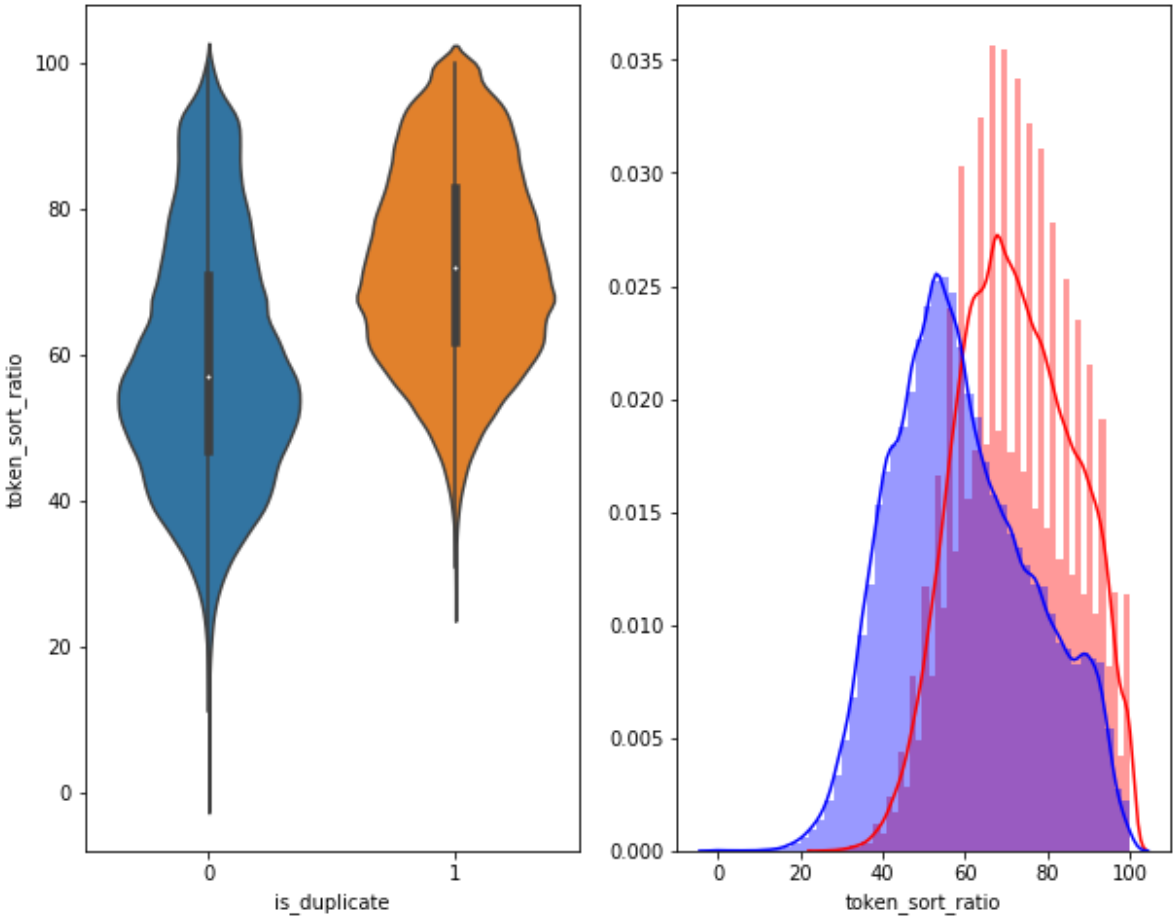
In [85]:

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



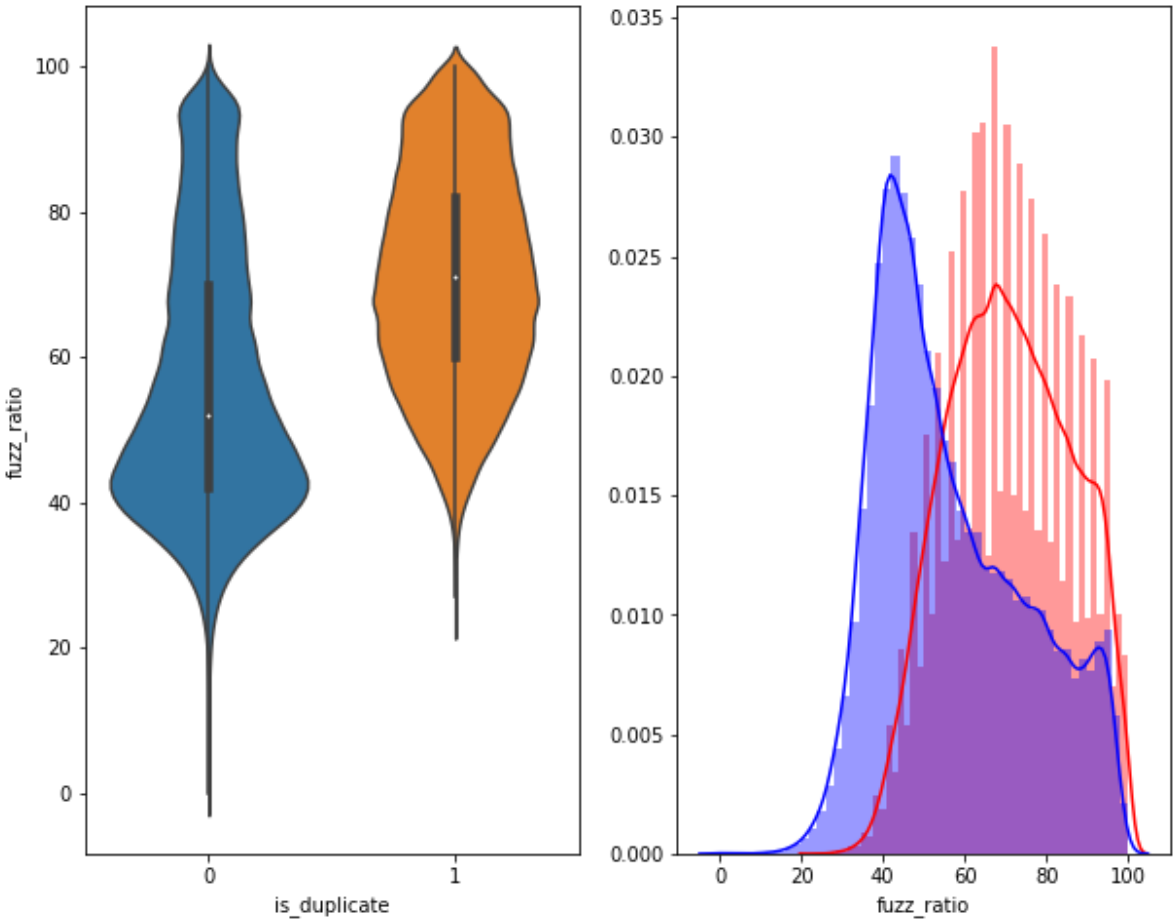
In [86]:

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

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

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



In [87]:

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

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

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



3.5.2 Visualization

In [88]:

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

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_
                                                 'last_word_eq', 'first_word_eq' , 'abs
                                                 'token_set_ratio' , 'token_sort_ratio'
                                                 'fuzz_partial_ratio' , 'longest_substr
y = dfp_subsampled['is_duplicate'].values
```

In [89]:

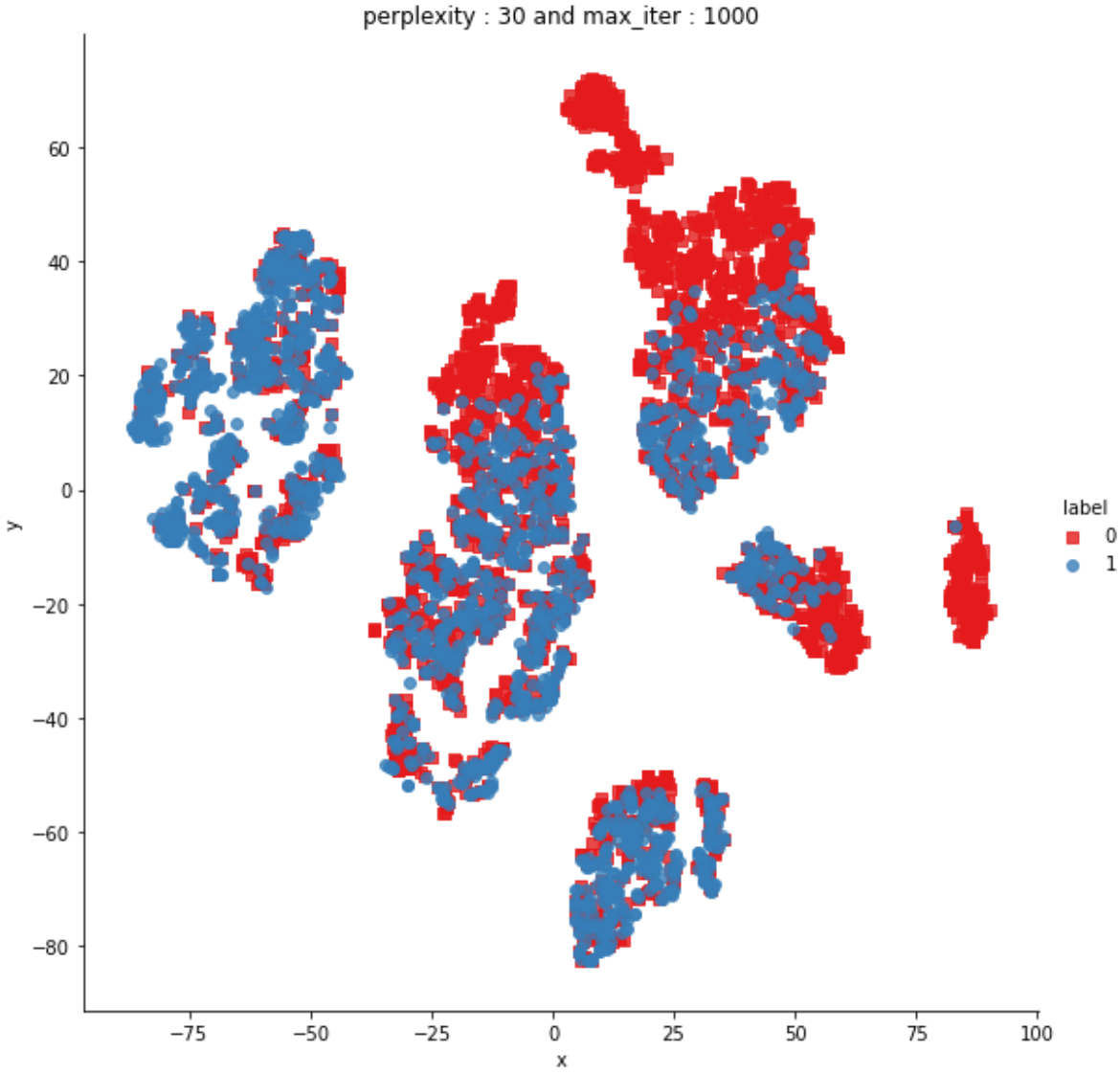
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.058s...
[t-SNE] Computed neighbors for 5000 samples in 0.565s...
[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: 0.116557
[t-SNE] Computed conditional probabilities in 0.373s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 it
erations in 4.088s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 i
terations in 2.812s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 i
terations in 2.894s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 i
terations in 2.864s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 i
terations in 2.920s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.273
346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 it
erations in 3.339s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 it
erations in 3.021s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 it
erations in 2.948s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 it
erations in 3.074s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 it
erations in 2.784s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 it
erations in 2.829s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 it
erations in 3.036s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 it
erations in 3.081s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 it
erations in 3.113s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 it
erations in 3.064s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 it
erations in 2.790s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 it
erations in 3.022s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 it
erations in 3.122s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 it
```

```
erations in 2.852s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 i
terations in 2.811s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [90]:

```
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 [91]:

```
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 0.484s...
[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: 0.116557
[t-SNE] Computed conditional probabilities in 0.277s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50
iterations in 13.728s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (50
iterations in 7.393s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (50
iterations in 8.901s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (50
iterations in 6.503s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (50
iterations in 6.364s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.7
29782
[t-SNE] Iteration 300: error = 1.4978341, gradient norm = 0.0006938 (50
iterations in 8.124s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50
iterations in 9.809s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50
iterations in 10.865s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50
iterations in 10.694s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50
iterations in 10.020s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50
iterations in 9.659s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50
iterations in 10.416s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50
iterations in 11.109s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50
iterations in 10.696s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50
iterations in 11.016s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50
iterations in 10.375s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50
iterations in 10.228s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50
iterations in 9.897s)
```

```
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50
iterations in 9.913s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (50
iterations in 9.945s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [92]:

```
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')
```

3.6 Featurizing text data with tfidf weighted word-vectors

In [95]:

```
pip install spacy
```

```
Collecting spacy
  Downloading https://files.pythonhosted.org/packages/69/d8/f3103202aeca
6fb0d2dbdd3a4ab1a7b86e9ad1d3cf8b23fa46bd466d64ac/spacy-2.2.3-cp37-cp37m-
win_amd64.whl (https://files.pythonhosted.org/packages/69/d8/f3103202aec
a6fb0d2dbdd3a4ab1a7b86e9ad1d3cf8b23fa46bd466d64ac/spacy-2.2.3-cp37-cp37m
-win_amd64.whl) (9.7MB)
Collecting srsly<1.1.0,>=0.1.0 (from spacy)
  Downloading https://files.pythonhosted.org/packages/2d/c2/2fdc6af49dea
dce26c6af0390bb29c0c2ad84d4df2784add1630b5ea18a5/srsly-1.0.0-cp37-cp37m-
win_amd64.whl (https://files.pythonhosted.org/packages/2d/c2/2fdc6af49de
adce26c6af0390bb29c0c2ad84d4df2784add1630b5ea18a5/srsly-1.0.0-cp37-cp37m
-win_amd64.whl) (179kB)
Collecting plac<1.2.0,>=0.9.6 (from spacy)
  Downloading https://files.pythonhosted.org/packages/86/85/40b8f66c2dd8
f4fd9f09d59b22720cffecf1331e788b8a0cab5bafb353d1/plac-1.1.3-py2.py3-none
-any.whl (https://files.pythonhosted.org/packages/86/85/40b8f66c2dd8f4fd
9f09d59b22720cffecf1331e788b8a0cab5bafb353d1/plac-1.1.3-py2.py3-none-an
y.whl)
Requirement already satisfied: numpy>=1.15.0 in c:\programdata\anaconda3
\lib\site-packages (from spacy) (1.16.5)
Collecting cymem<2.1.0,>=2.0.2 (from spacy)
  Downloading https://files.pythonhosted.org/packages/84/d1/35eab0c8cc9f
d9432becaf3e90144762b3201a45079e62c47a8ae8739763/cymem-2.0.3-cp37-cp37m-
win_amd64.whl (https://files.pythonhosted.org/packages/84/d1/35eab0c8cc9
fd9432becaf3e90144762b3201a45079e62c47a8ae8739763/cymem-2.0.3-cp37-cp37m
-win_amd64.whl)
Collecting catalogue<1.1.0,>=0.0.7 (from spacy)
  Downloading https://files.pythonhosted.org/packages/4b/4c/0e0fa8b1e193
c1e09a6b72807ff4ca17c78f68f0c0f4459bc8043c66d649/catalogue-0.2.0-py2.py3
-none-any.whl (https://files.pythonhosted.org/packages/4b/4c/0e0fa8b1e19
3c1e09a6b72807ff4ca17c78f68f0c0f4459bc8043c66d649/catalogue-0.2.0-py2.py
3-none-any.whl)
Collecting wasabi<1.1.0,>=0.4.0 (from spacy)
  Downloading https://files.pythonhosted.org/packages/9b/f5/01bc4156ac46
c46297b6291ab438336ff66c096bd37e85f7381e6254f908/wasabi-0.5.0-py3-none-a
ny.whl (https://files.pythonhosted.org/packages/9b/f5/01bc4156ac46c46297
b6291ab438336ff66c096bd37e85f7381e6254f908/wasabi-0.5.0-py3-none-any.wh
l)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\programdata
\anaconda3\lib\site-packages (from spacy) (2.22.0)
Collecting preshed<3.1.0,>=3.0.2 (from spacy)
  Downloading https://files.pythonhosted.org/packages/3c/5a/0d1b575ed409
89d74fab25723083837c220246b25f3582917135cb32453f/preshed-3.0.2-cp37-cp37
m-win_amd64.whl (https://files.pythonhosted.org/packages/3c/5a/0d1b575ed
40989d74fab25723083837c220246b25f3582917135cb32453f/preshed-3.0.2-cp37-c
p37m-win_amd64.whl) (105kB)
Collecting murmurhash<1.1.0,>=0.28.0 (from spacy)
  Downloading https://files.pythonhosted.org/packages/4f/7b/d77bc9bb101e
113884b2d70a118e7ec8dcc9846a35a0e10d47ca37acdcbf/murmurhash-1.0.2-cp37-c
p37m-win_amd64.whl (https://files.pythonhosted.org/packages/4f/7b/d77bc9
bb101e113884b2d70a118e7ec8dcc9846a35a0e10d47ca37acdcbf/murmurhash-1.0.2-
cp37-cp37m-win_amd64.whl)
Collecting thinc<7.4.0,>=7.3.0 (from spacy)
  Downloading https://files.pythonhosted.org/packages/9e/ed/7edded747247
47f7dfc513f85b483db7828e4a1ed072c9625188dcb633a5/thinc-7.3.1-cp37-cp37m-
win_amd64.whl (https://files.pythonhosted.org/packages/9e/ed/7edded74724
```

```
747f7dfc513f85b483db7828e4a1ed072c9625188dcb633a5/thinc-7.3.1-cp37-cp37m
-win_amd64.whl) (2.0MB)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\li
b\site-packages (from spacy) (41.4.0)
Collecting blis<0.5.0,>=0.4.0 (from spacy)
  Downloading https://files.pythonhosted.org/packages/d5/7e/1981d5389b75
543f950026de40a9d346e2aec7e860b2800e54e65bd46c06/blis-0.4.1-cp37-cp37m-w
in_amd64.whl (https://files.pythonhosted.org/packages/d5/7e/1981d5389b75
543f950026de40a9d346e2aec7e860b2800e54e65bd46c06/blis-0.4.1-cp37-cp37m-w
in_amd64.whl) (5.0MB)
Requirement already satisfied: importlib-metadata>=0.20; python_version
 < "3.8" in c:\programdata\anaconda3\lib\site-packages (from catalogue<
1.1.0,>=0.0.7->spacy) (0.23)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\programdata\a
naconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 i
n c:\programdata\anaconda3\lib\site-packages (from requests<3.0.0,>=2.1
3.0->spacy) (1.24.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anac
onda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2019.9.1
1)
Requirement already satisfied: idna<2.9,>=2.5 in c:\programdata\anaconda
3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2.8)
Requirement already satisfied: tqdm<5.0.0,>=4.10.0 in c:\programdata\ana
conda3\lib\site-packages (from thinc<7.4.0,>=7.3.0->spacy) (4.36.1)
Requirement already satisfied: zipp>=0.5 in c:\programdata\anaconda3\lib
\site-packages (from importlib-metadata>=0.20; python_version < "3.8"->c
atalogue<1.1.0,>=0.0.7->spacy) (0.6.0)
Requirement already satisfied: more-itertools in c:\programdata\anaconda
3\lib\site-packages (from zipp>=0.5->importlib-metadata>=0.20; python_ve
rsion < "3.8"->catalogue<1.1.0,>=0.0.7->spacy) (7.2.0)
Installing collected packages: srsly, plac, cymem, catalogue, wasabi, mu
rmurhash, preshed, blis, thinc, spacy
Successfully installed blis-0.4.1 catalogue-0.2.0 cymem-2.0.3 murmurhash
-1.0.2 plac-1.1.3 preshed-3.0.2 spacy-2.2.3 srsly-1.0.0 thinc-7.3.1 wasa
bi-0.5.0
Note: you may need to restart the kernel to use updated packages.
```

In [96]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as npx
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

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

In [97]:

```
# avoid decoding problems
df = pd.read_csv("train.csv")

# 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))
```

In [98]:

```
df.head()
```

Out[98]:

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

In [99]:

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

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

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

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

In [117]:

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
# cite :https://github.com/hamelsmu/Seq2Seq_Tutorial/issues/1
nlp = spacy.load('en')

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

```
100%|██████████████████████████████████████████████████████████████████████
███| 404290/404290 [1:39:12<00:00, 67.92it/s]
```

In [120]:

```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = 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%|██████████████████████████████████████████████████████████████████████
███| 404290/404290 [1:39:51<00:00, 67.48it/s]
```

In [121]:

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

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous no
```

In [122]:

```
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)
```

In [123]:

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

Out[123]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0

In [124]:

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

Out[124]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total
0	0	1	1	66	57	14	12	10.0	23.0
1	1	4	1	51	88	8	13	4.0	20.0
2	2	1	1	73	59	14	10	4.0	24.0
3	3	1	1	50	65	11	9	0.0	19.0
4	4	3	1	76	39	13	7	2.0	20.0

In [125]:

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

Out[125]:

	0	1	2	3	4	5	6	
0	-6.179507	37.450731	-67.929894	32.224274	143.348826	135.374574	17.865208	54.5623
1	9.236668	-80.371416	-45.785907	78.291656	183.568221	100.894077	74.344804	48.3608
2	97.546832	22.972194	-39.558379	18.723413	56.928618	48.307643	8.719268	36.8937
3	57.586978	-22.017089	-4.599294	-88.939271	-4.732171	-54.209048	74.614947	106.5337
4	83.185784	-40.506985	-83.403923	-52.648658	79.074884	-19.038248	53.728722	97.6486

5 rows × 96 columns

In [126]:

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

Out[126]:

	0	1	2	3	4	5	6	
0	-14.616981	59.755488	-53.263745	19.514497	113.916473	101.657056	8.561499	66.232
1	-3.565742	-16.844571	-130.911785	0.320254	79.350278	23.562028	79.124551	84.119
2	156.833630	59.991896	-8.414311	29.251426	133.680218	112.457566	89.849781	21.613
3	41.472439	56.717317	31.530616	-5.520164	33.454800	79.596179	15.508996	40.042
4	-14.446975	-4.338255	-70.196208	-48.636382	18.356858	-50.807069	24.311196	60.043

5 rows × 96 columns

In [127]:

```
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.shap
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 96
Number of features in question2 w2v  dataframe : 96
Number of features in final dataframe  : 221
```

In [128]:

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

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [142]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
#from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
from datetime import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier

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

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
import spacy
from tqdm import tqdm
from datetime import datetime as dt
```

In [141]:

```
pip install mlxtend
```

```
Collecting mlxtend
  Downloading https://files.pythonhosted.org/packages/52/04/c362f34f666f0d
dc7cf593805e64d64fa670ed96fd9302e68549dd48287d/mlxtend-0.17.0-py2.py3-none
-any.whl (https://files.pythonhosted.org/packages/52/04/c362f34f666f0ddc7c
f593805e64d64fa670ed96fd9302e68549dd48287d/mlxtend-0.17.0-py2.py3-none-an
y.whl) (1.3MB)
Requirement already satisfied: numpy>=1.16.2 in c:\programdata\anaconda3\l
ib\site-packages (from mlxtend) (1.16.5)
Requirement already satisfied: scikit-learn>=0.20.3 in c:\programdata\anac
onda3\lib\site-packages (from mlxtend) (0.21.3)
Requirement already satisfied: scipy>=1.2.1 in c:\programdata\anaconda3\li
b\site-packages (from mlxtend) (1.3.1)
Requirement already satisfied: matplotlib>=3.0.0 in c:\programdata\anacond
a3\lib\site-packages (from mlxtend) (3.1.1)
Requirement already satisfied: joblib>=0.13.2 in c:\programdata\anaconda3
\lib\site-packages (from mlxtend) (0.13.2)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib
\site-packages (from mlxtend) (41.4.0)
Requirement already satisfied: pandas>=0.24.2 in c:\programdata\anaconda3
\lib\site-packages (from mlxtend) (0.25.1)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxten
d) (2.4.2)
Requirement already satisfied: python-dateutil>=2.1 in c:\programdata\anac
onda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.0)
Requirement already satisfied: pytz>=2017.2 in c:\programdata\anaconda3\li
b\site-packages (from pandas>=0.24.2->mlxtend) (2019.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-pa
ckages (from cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.12.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.17.0
Note: you may need to restart the kernel to use updated packages.
```

In [129]:

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

In [130]:

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

Out[130]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	las
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	

3 rows × 221 columns

4.2 Random train test split

In [131]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,data['is_duplicate'],stratify=
                                                    random_state=32)
```

In [132]:

```
print("Shape of X_train :",X_train.shape)
print("shape of y train :", y_train.shape)
```

```
Shape of X_train : (37500, 221)
shape of y train : (37500,)
```

In [133]:

```
X_train.head()
```

Out[133]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_
23561	23561	23561	0	0.333322	0.333322	0.749981	0.428565	0.571420	0.39
3536	3536	3536	0	0.000000	0.000000	0.333322	0.166664	0.111110	0.08
33192	33192	33192	1	0.666656	0.399996	0.749981	0.374995	0.699993	0.38
35725	35725	35725	0	0.999950	0.666644	0.000000	0.000000	0.399992	0.28
6320	6320	6320	0	0.749981	0.599988	0.749981	0.599988	0.749991	0.59

5 rows × 221 columns

In [134]:

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

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

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

```
Number of data points in train data : (37500, 218)
Number of data points in test data : (12500, 218)
y train shape : (37500,)
```

In [ ]:

```
from collections import Counter
from collections import Counter, defaultdict
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[1])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

4.3 def Confustion Matrix

In [143]:

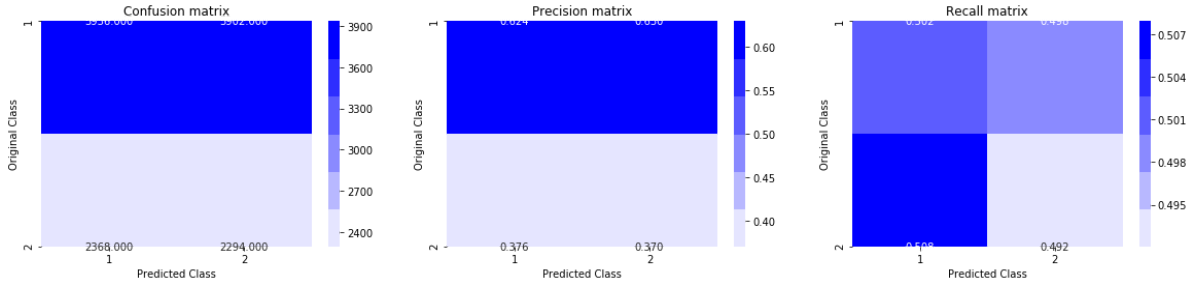
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predic

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
    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=la
    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=la
    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 [144]:

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

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

```
Log loss on Test Data using Random Model 0.8892749522870612
```

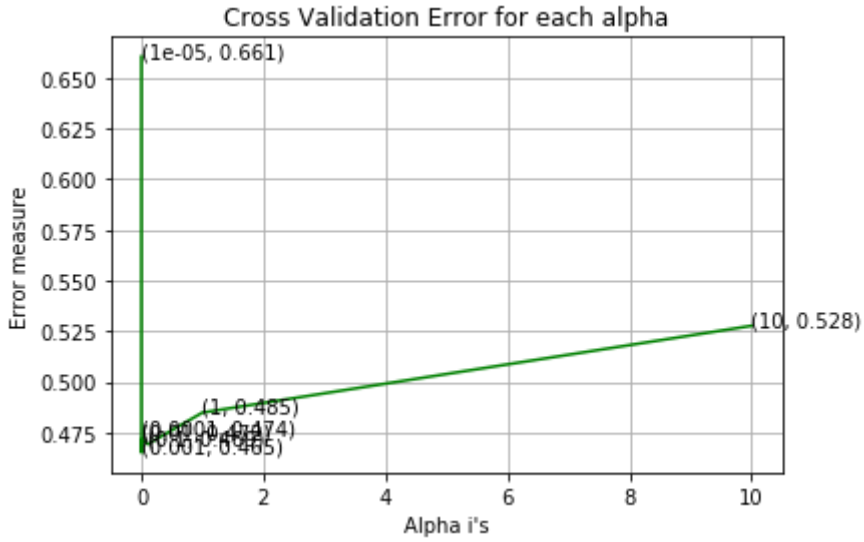


In [145]:

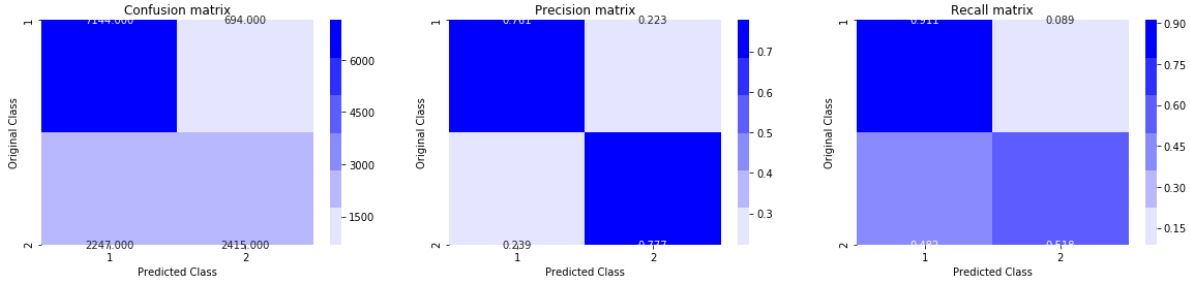
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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




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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(X_test)
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-05 The log loss is: 0.6605122740522953
For values of alpha =  0.0001 The log loss is: 0.47394741360516257
For values of alpha =  0.001 The log loss is: 0.46503561049739617
For values of alpha =  0.01 The log loss is: 0.47178383668593543
For values of alpha =  0.1 The log loss is: 0.4691746719527075
```

```
For values of alpha =  1 The log loss is: 0.4848014377495121
For values of alpha =  10 The log loss is: 0.5276390243996072
```




```
For values of best alpha =  0.001 The train log loss is: 0.458661050752141
5
For values of best alpha =  0.001 The test log loss is: 0.4650356104973961
7
Total number of data points : 12500
```



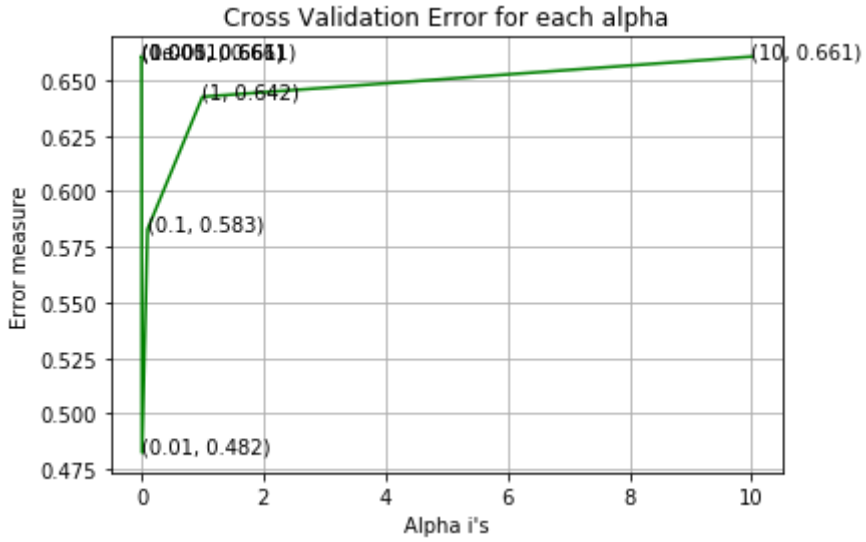
4.5 Linear SVM with Hyperparameter Tuning

In [146]:

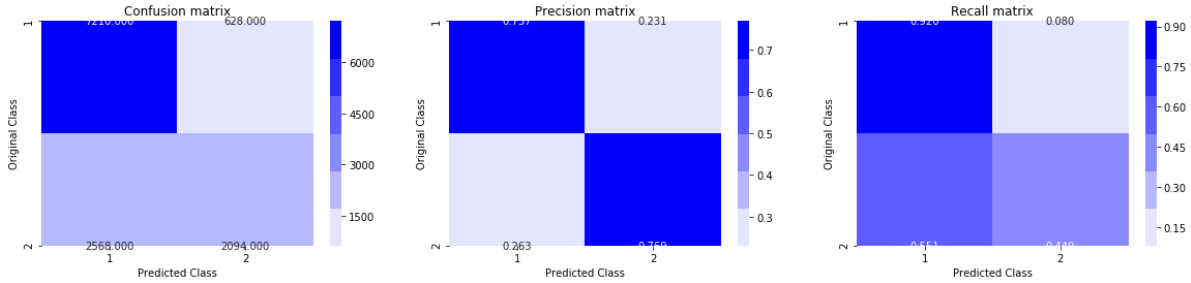
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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




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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(X_test)
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-05 The log loss is: 0.6605122740522953
For values of alpha =  0.0001 The log loss is: 0.6605122740522953
For values of alpha =  0.001 The log loss is: 0.6605122740522953
For values of alpha =  0.01 The log loss is: 0.4824487537615871
For values of alpha =  0.1 The log loss is: 0.5829063073648578
```

```
For values of alpha =  1 The log loss is: 0.6424638067389172
For values of alpha =  10 The log loss is: 0.6605122740522953
```




```
For values of best alpha =  0.01 The train log loss is: 0.4803502509804249
4
For values of best alpha =  0.01 The test log loss is: 0.4824487537615871
Total number of data points : 12500
```



4.6 XGBoost on Random model

In [152]:

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

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

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

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

xgdmat = xgb.DMatrix(X_train,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
```

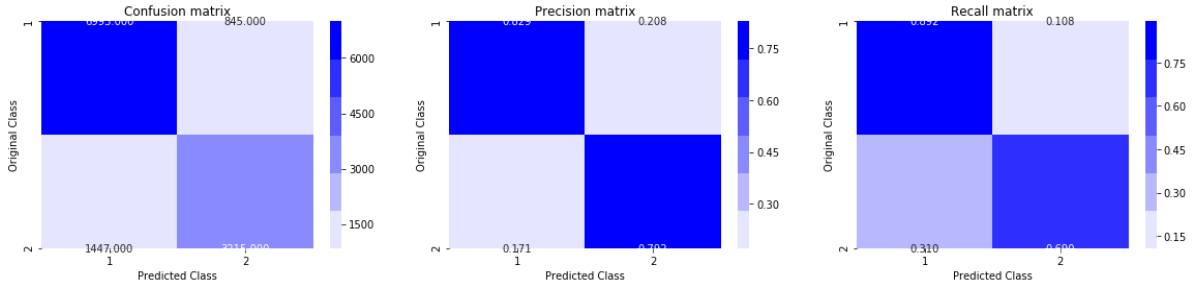
```
[0]     train-logloss:0.684831  valid-logloss:0.68492
Multiple eval metrics have been passed: 'valid-logloss' will be used for
early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.614815  valid-logloss:0.615143
[20]    train-logloss:0.564044  valid-logloss:0.564804
[30]    train-logloss:0.525747  valid-logloss:0.526968
[40]    train-logloss:0.495944  valid-logloss:0.497558
[50]    train-logloss:0.472833  valid-logloss:0.474814
[60]    train-logloss:0.454838  valid-logloss:0.457093
[70]    train-logloss:0.440087  valid-logloss:0.442566
[80]    train-logloss:0.428034  valid-logloss:0.430735
[90]    train-logloss:0.418248  valid-logloss:0.421226
[100]   train-logloss:0.410244  valid-logloss:0.413337
[110]   train-logloss:0.403465  valid-logloss:0.406766
[120]   train-logloss:0.397676  valid-logloss:0.401171
[130]   train-logloss:0.392861  valid-logloss:0.396535
[140]   train-logloss:0.388401  valid-logloss:0.392196
[150]   train-logloss:0.384944  valid-logloss:0.389072
[160]   train-logloss:0.381833  valid-logloss:0.386256
[170]   train-logloss:0.379131  valid-logloss:0.383879
[180]   train-logloss:0.376806  valid-logloss:0.381937
[190]   train-logloss:0.374439  valid-logloss:0.379917
[200]   train-logloss:0.372371  valid-logloss:0.378125
[210]   train-logloss:0.370508  valid-logloss:0.376647
[220]   train-logloss:0.368713  valid-logloss:0.375195
[230]   train-logloss:0.367022  valid-logloss:0.373822
[240]   train-logloss:0.365166  valid-logloss:0.372401
[250]   train-logloss:0.363515  valid-logloss:0.371158
[260]   train-logloss:0.361935  valid-logloss:0.370026
[270]   train-logloss:0.360359  valid-logloss:0.369001
[280]   train-logloss:0.358714  valid-logloss:0.367716
[290]   train-logloss:0.35735   valid-logloss:0.366786
[300]   train-logloss:0.356012  valid-logloss:0.365899
[310]   train-logloss:0.354544  valid-logloss:0.364884
[320]   train-logloss:0.353221  valid-logloss:0.364036
[330]   train-logloss:0.351998  valid-logloss:0.36332
[340]   train-logloss:0.35076   valid-logloss:0.362538
[350]   train-logloss:0.34951   valid-logloss:0.361727
[360]   train-logloss:0.34822   valid-logloss:0.360976
```

```
[370]   train-logloss:0.347083  valid-logloss:0.360373
[380]   train-logloss:0.345999  valid-logloss:0.359844
[390]   train-logloss:0.344957  valid-logloss:0.359314
[399]   train-logloss:0.343868  valid-logloss:0.358692
The test log loss is: 0.3586922006776975
```

In [153]:

```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
Total number of data points : 12500
```



1. Let us Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

5.0 Reading Data from file

In [154]:

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

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

In [155]:

```
df1.head()
```

Out[155]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988
2	2	5	6	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0	0.399992	0.333328	0.399992	0.249997
3	3	7	8	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0	0.000000	0.000000	0.000000	0.000000
4	4	9	10	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.666644

5 rows × 21 columns

In [156]:

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

In [157]:

```
dfnlp.head()
```

Out[157]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988
2	2	5	6	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0	0.399992	0.333328	0.399992	0.249997
3	3	7	8	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0	0.000000	0.000000	0.000000	0.000000
4	4	9	10	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.666644

5 rows × 32 columns

In [158]:

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

```
          id   qid1   qid2                                               questi
on1  \
3306    3306   6553   6554
NaN
13016  13016  25026  25027
NaN
20072  20072  37898  37899                             how could i solve th
is
20794  20794  39204  39205
NaN
47056  47056  84067  84068  is there anywhere in the world offering pain
m...

                                              question2  is_duplicate  \
3306   why is cornell own endowment the lowest in the...             0
13016                 why should one not work at google             0
20072                                               NaN             0
20794  what is the gmail tech support help phone number             0
47056                                               NaN             0

       cwc_min  cwc_max  csc_min  csc_max  ...  freq_qid2  q1len  q2len  \
3306       0.0      0.0      0.0      0.0  ...          1      1     56
13016      0.0      0.0      0.0      0.0  ...          2      1     34
20072      0.0      0.0      0.0      0.0  ...          2     23      6
20794      0.0      0.0      0.0      0.0  ...          1      1     49
47056      0.0      0.0      0.0      0.0  ...          1    117      1

       q1_n_words  q2_n_words  word_Common  word_Total  word_share  \
3306            1          10          0.0        10.0         0.0
13016           1           7          0.0         8.0         0.0
20072           5           1          0.0         6.0         0.0
20794           1           9          0.0        10.0         0.0
47056          19           1          0.0        19.0         0.0

       freq_q1+q2  freq_q1-q2
3306            2           0
13016           4           0
20072           4           0
20794           2           0
47056           4           2

[5 rows x 32 columns]
```

In [159]:

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

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate, cwc_min, cwc
_max, csc_min, csc_max, ctc_min, ctc_max, last_word_eq, first_word_eq, abs
_len_diff, mean_len, token_set_ratio, token_sort_ratio, fuzz_ratio, fuzz_p
artial_ratio, longest_substr_ratio, freq_qid1, freq_qid2, q1len, q2len, q1
_n_words, q2_n_words, word_Common, word_Total, word_share, freq_q1+q2, fre
q_q1-q2]
Index: []

[0 rows x 32 columns]
```

5.1 Splitting data Train, Test , CV

In [161]:

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

In [164]:

```
X_test= X_test.drop(('is_duplicate'),axis=1)
print("Shape of x_test :", X_test.shape)
```

```
Shape of x_test : (12500, 31)
```

In [165]:

```
print("shape of y train :", y_train.shape)
print("Shape of y test :", y_test.shape)
```

```
shape of y train : (37500,)
Shape of y test : (12500,)
```

In [166]:

```
X_train.head(1)
```

Out[166]:

	id	qid1	qid2	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ct
23561	23561	44124	44125	how do i learn geography for nda	how do i learn to accept myself and my appeara...	0.333322	0.333322	0.749981	0.428565	0

1 rows × 31 columns

5.2 TFIDF vectorizer on Questions Text Dat

In [167]:

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

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


vectorizer.fit(questions)
```

Out[167]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=
None,
                min_df=10, ngram_range=(1, 2), norm='l2', preprocessor=Non
e,
                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)
```

5.3 TFIDF Vectorizer on Train data _ question1 and question2

In [168]:

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

print("the number of unique words in tfidf_train_ques1: ", tfidf_train_ques1.get_shape(
#question2
tfidf_train_ques2= vectorizer.transform(X_train['question2'])
print("Shape of matrix after one hot encodig tfidf_train_ques2: ",tfidf_train_ques2.sha
print("the number of unique words in tfidf_train_ques2: ", tfidf_train_ques2.get_shape(
```

```
Shape of matrix after one hot encodig tfidf_train_ques1:  (37500, 13365)
the number of unique words in tfidf_train_ques1:  13365
Shape of matrix after one hot encodig tfidf_train_ques2:  (37500, 13365)
the number of unique words in tfidf_train_ques2:  13365
```

In [169]:

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

In [171]:

```
X_train_feature_df.head()
```

Out[171]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_e
23561	0.333322	0.333322	0.749981	0.428565	0.571420	0.399996	0.0	1
3536	0.000000	0.000000	0.333322	0.166664	0.111110	0.083333	0.0	0
33192	0.666656	0.399996	0.749981	0.374995	0.699993	0.388887	0.0	1
35725	0.999950	0.666644	0.000000	0.000000	0.399992	0.285710	1.0	0
6320	0.749981	0.599988	0.749981	0.599988	0.749991	0.599994	1.0	0

5 rows × 26 columns

In [172]:

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

#so we need to convert our feature data into sparse matrix so that we will combine our
train_feat_sparse = scipy.sparse.csr_matrix(X_train_feature_df)

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

```
train Shape Before ->  (37500, 26)  Type <class 'pandas.core.frame.DataFra
me'>
train Shape After->  (37500, 26)  Type <class 'scipy.sparse.csr.csr_matri
x'>
```

In [173]:

```
# Test data question1
tfidf_test_ques1= vectorizer.transform(X_test['question1'])
print("Shape of matrix after one hot encodig tfidf_test_ques1:",tfidf_test_ques1.shape)
print("the number of unique words in tfidf_test_ques1: ", tfidf_test_ques1.get_shape()[

#Test data question2
tfidf_test_ques2= vectorizer.transform(X_test['question2'])
print("Shape of matrix after one hot encodig tfidf_test_ques2: ",tfidf_test_ques2.shape
print("the number of unique words in tfidf_test_ques2: ", tfidf_test_ques2.get_shape()[
```

```
Shape of matrix after one hot encodig tfidf_test_ques1: (12500, 13365)
the number of unique words in tfidf_test_ques1:  13365
Shape of matrix after one hot encodig tfidf_test_ques2:  (12500, 13365)
the number of unique words in tfidf_test_ques2:  13365
```

In [174]:

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

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

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

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

```
test Shape Before ->  (12500, 26)  Type <class 'pandas.core.frame.DataFram
e'>
test Shape After->  (12500, 26)  Type <class 'scipy.sparse.csr.csr_matri
x'>
```

In [175]:

```
# combining our tfidf and features into one

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


tfidf_train = hstack((tfidf_train_ques1,tfidf_train_ques2))

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

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

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

```
train data shape (37500, 26730)
Test data shape  (12500, 26730)
```

In [176]:

```
tfidf_train.shape
```

Out[176]:

```
(37500, 26730)
```

In [177]:

```
from scipy.sparse import hstack

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

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

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

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

```
train data shape (37500, 26756)
Test data shape  (12500, 26756)
```

In [178]:

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

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

```
Final Shape of the Data matrix
(37500, 26756) (37500,)
(12500, 26756) (12500,)
```

In [179]:

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

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

5.4 def Confusion Matrix

In [180]:

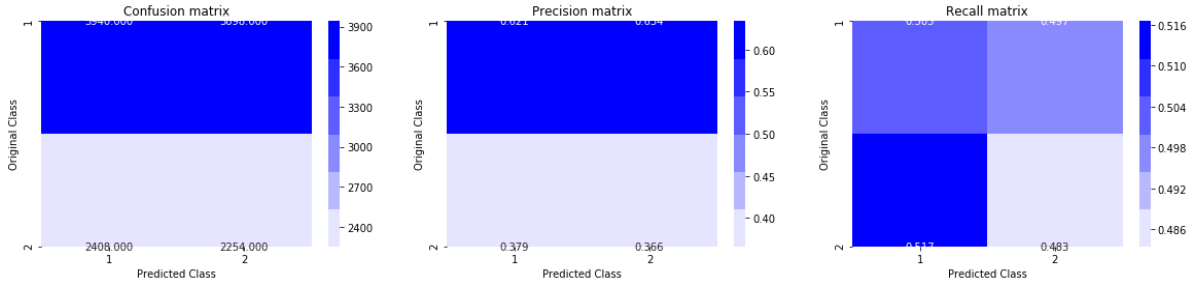
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predic

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
    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=la
    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=la
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

5.5 Building Random model (worse case log loss) - TFIDF

In [181]:

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

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

```
Log loss on Test Data using Random Model 0.8839826362854141
```

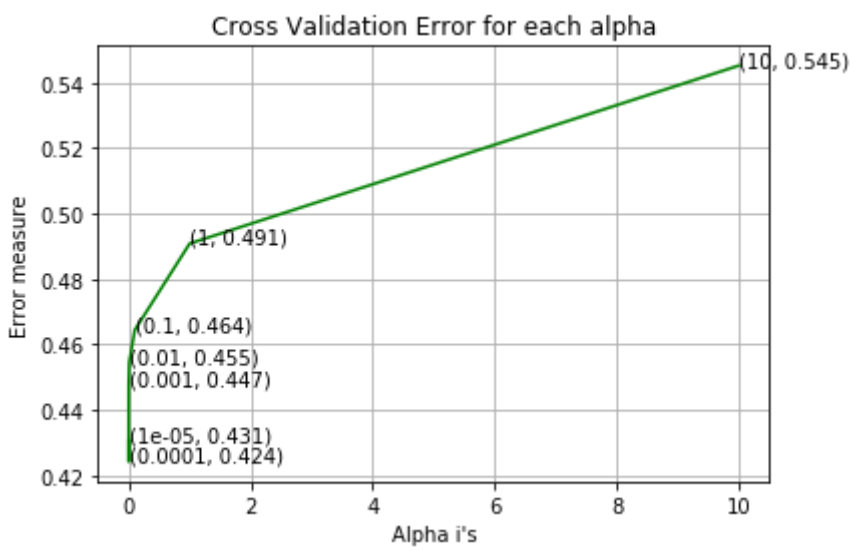


5.6 Logistic Regression with hyperparameter tuning

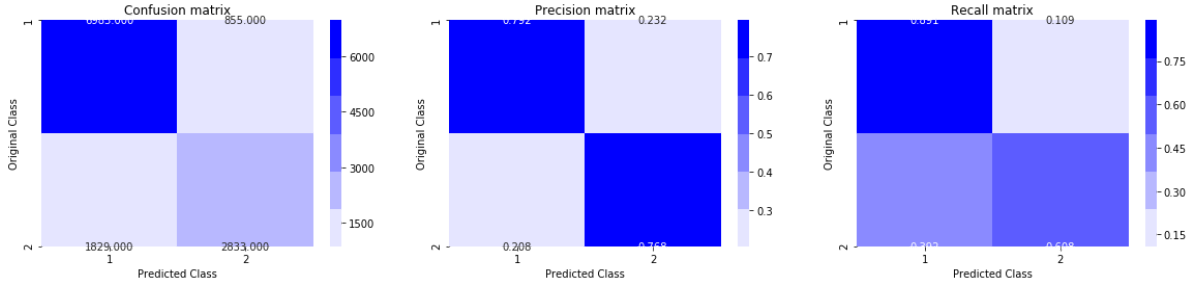
In [182]:

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

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

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


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

predict_y = sig_clf.predict_proba(tfidf_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(tfidf_test)
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-05 The log loss is: 0.43057045667853533
For values of alpha =  0.0001 The log loss is: 0.42416550170315664
For values of alpha =  0.001 The log loss is: 0.44742936691017465
For values of alpha =  0.01 The log loss is: 0.45458433860368086
For values of alpha =  0.1 The log loss is: 0.4643584995447944
For values of alpha =  1 The log loss is: 0.49080407521050934
For values of alpha =  10 The log loss is: 0.5451775161339065
```




```
For values of best alpha =  0.0001 The train log loss is: 0.41338058044966
347
For values of best alpha =  0.0001 The test log loss is: 0.424165501703156
64
Total number of data points : 12500
```

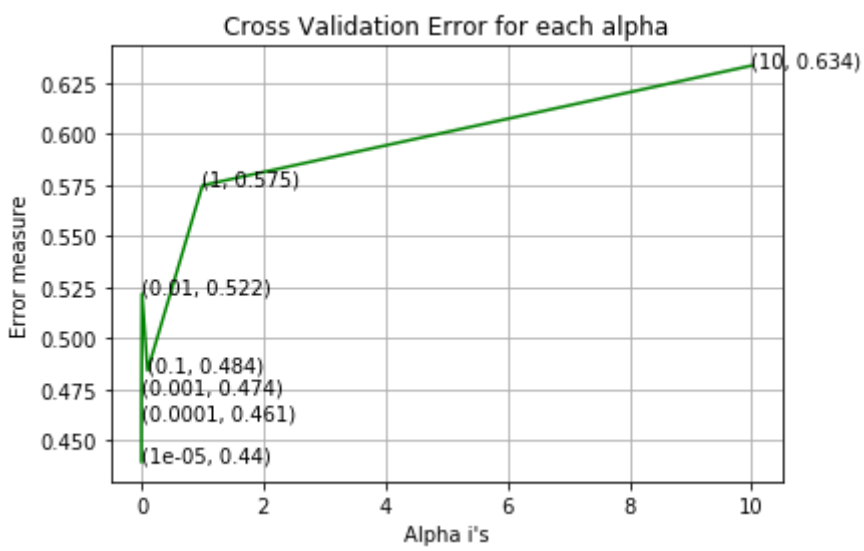


5.7 Linear SVM Hyperparameter Tuning

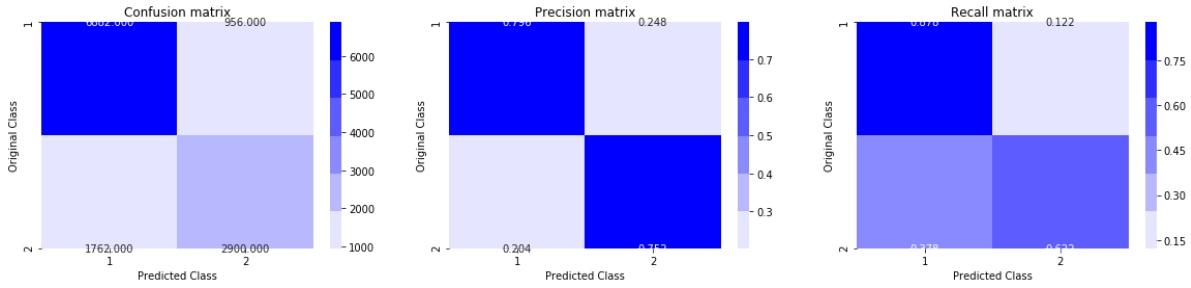
In [183]:

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

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

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



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

predict_y = sig_clf.predict_proba(tfidf_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(tfidf_test)
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-05 The log loss is: 0.43971930193051834
For values of alpha =  0.0001 The log loss is: 0.4609187140027068
For values of alpha =  0.001 The log loss is: 0.4735602155265985
For values of alpha =  0.01 The log loss is: 0.5219821811005141
For values of alpha =  0.1 The log loss is: 0.4843230596967703
For values of alpha =  1 The log loss is: 0.5749588676366658
For values of alpha =  10 The log loss is: 0.6335853743705856
```




```
For values of best alpha =  1e-05 The train log loss is: 0.422889829583406
6
For values of best alpha =  1e-05 The test log loss is: 0.4397193019305183
4
Total number of data points : 12500
```



XGBoost - Hyperparameter Tuning

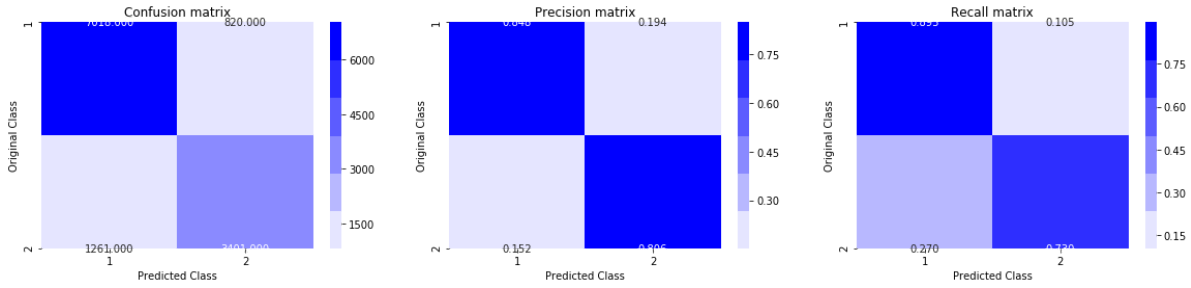
In [185]:

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




params = {

        "learning_rate":sc.uniform(0.05,0.3),
        'max_depth': sc.randint(3,15),
        'n_estimators' : sc.randint(10,200),
        "min_child_weight" : [ 1, 3, 5, 7 ],
        'gamma': sc.uniform(0.0,0.5)
        }
x_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss',n_jobs=-

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

#xgb_random_search.fit(X_train, y_train)


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

In [190]:

```
# best params
bst = xgb.XGBClassifier(max_depth=10,learning_rate=0.1042,objective='binary:logistic',g
bst.fit(tfidf_train, y_train)


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

predict_y = clf_calib.predict_proba(tfidf_train)

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

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

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

```
The train log loss is:  0.2506374948791325
The test log loss is :  0.3456727415454542
```



TFIDF W2V 50 K Data

In [191]:

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

dftw_50k.head()
```

```
Columns in dftw_50k dataframe:

Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_wor
ds',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q
2'],
      dtype='object')
```

Out[191]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len
292007	292007	413533	413534	Does any book describe the ancient Indian Guru...	Where can I read about Gurukul system of educa...	1	1	1	67
297733	297733	141532	46839	What are other question-asking websites like Q...	Are there any websites that has similar functi...	1	9	5	51
159027	159027	248319	248320	Which country is the least intolerant towards ...	Can this be the most viewed question on Quora?	0	1	1	56
287891	287891	169424	408732	Where can I learn HTML and CSS?	Where can I learn more about HTML and CSS?	1	3	1	31
165500	165500	184440	256987	Do men prefer women with no pubic hair, landin...	Guys, do you prefer a woman's pubic hair to be...	1	1	1	71

In [192]:

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

In [193]:

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

In [194]:

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

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

In [195]:

```
print("Shape of x tw train data:", x_train_tw.shape)
print("Shape of xtw test data:", x_test_tw.shape)
print("Shape of y tw train data:", y_train_tw.shape)
print("Shape of y tw test data:", y_test_tw.shape)
```

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

In [196]:

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

# Importing library
from sklearn.feature_extraction.text import TfidfVectorizer

# Merge texts
questions = list(x_train_tw['question1']) + list(x_train_tw['question2'])

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

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

In [201]:

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

# Importing libraries
import os
import spacy
from tqdm import tqdm

def vec(xtw):

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

    vecs = []

    # https://github.com/noamraph/tqdm
    # tqdm is used to print the progress bar
    for qu in tqdm(list(xtw)):

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

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

            try:
                idf = word2tfidf[str(word)]

            except:
                idf = 0

            # compute final vec
            mean_vec += vec * idf

        mean_vec = mean_vec.mean(axis = 0)
        vecs.append(mean_vec)
    #dftw_100k['q1_feats_m'] = list(vecs1)

    return vecs
```

In [202]:

```
# Calling 'vec' function for train question1
x_train_tw['que1_tw'] = vec(x_train_tw['question1'])
# Calling 'vec' function for train question2
x_train_tw['que2_tw'] = vec(x_train_tw['question2'])
# Calling 'vec' function for test question1
x_test_tw['que1_tw'] = vec(x_test_tw['question1'])
# Calling 'vec' function for test question2
x_test_tw['que2_tw'] = vec(x_test_tw['question2'])
```

```
100%|████████████████████████████████████████████████████████████████████████
███████| 35000/35000 [07:55<00:00, 73.61it/s]
100%|████████████████████████████████████████████████████████████████████████
███████| 35000/35000 [07:43<00:00, 75.48it/s]
100%|████████████████████████████████████████████████████████████████████████
███████| 15000/15000 [03:18<00:00, 75.57it/s]
100%|████████████████████████████████████████████████████████████████████████
███████| 15000/15000 [03:10<00:00, 78.60it/s]
```

In [204]:

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

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


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

```
Type of x_train_tw['que1_tw']: <class 'pandas.core.series.Series'>
Type of x_train_tw['que2_tw']: <class 'pandas.core.series.Series'>
Type of x_test_tw['que1_tw'] : <class 'pandas.core.series.Series'>
Type of x_test_tw['que2_tw'] : <class 'pandas.core.series.Series'>
Shape of x train question1    : (35000,)
Shape of x test question1 data: (15000,)
Shape of x train question2    : (35000,)
Shape of x test question2 data: (15000,)
```

In [205]:

```
# Train dataframe

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

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


# Test dataframe

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

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

In [206]:

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

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

# Dropping question1 and question2 columns from final_test dataframe

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

In [207]:

```
final_tr_tw = final_tr_tw.fillna(0) # Filling train dataframe
final_te_tw = final_te_tw.fillna(0) # Filling test dataframe

# Dropping question1 and question2 columns from final_train dataframe
final_tr_tw = final_tr_tw.drop(['question1', 'question2', 'que1_tw', 'que2_tw'], axis =
# Dropping question1 and question2 columns from final_test dataframe
final_te_tw = final_te_tw.drop(['question1', 'question2', 'que1_tw', 'que2_tw'], axis =
```

In [208]:

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

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

```
Shape of final_tr_tw dataframe: (35000, 205)

Shape of final_te_tw dataframe: (15000, 205)
```

In [209]:

```
# Saving final train data
final_tr_tw.to_csv("quora_final_tr_tw.csv")

# Saving final test data
final_te_tw.to_csv("quora_final_te_tw.csv")
```

In [210]:

```
# Import Libraries
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
from sklearn.metrics import log_loss

start = dt.now()

# Parameters we need to try are
param_grid = {'n_estimators' : [5, 10, 100, 500], 'max_depth' : [2, 5, 8, 10]}

rs_k = RandomizedSearchCV(estimator = XGBClassifier(objective = 'binary:logistic', eval
                          param_distributions = param_grid)

# fit train sets
rs_k.fit(final_tr_tw, y_train_tw)

# Prediction
predict_tw = rs_k.predict(final_te_tw)

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

```
Time taken to run this cell: 1:19:23.117799
```

In [211]:

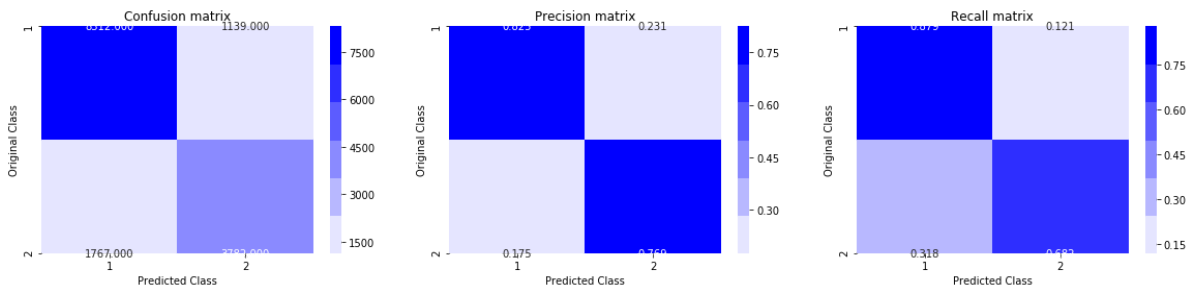
```
bp = rs_k.best_params_
bs = rs_k.best_score_

print("Optimal hyperParameter:", bp, '\n')
print("Maximum accuracy:", bs * 100)
```

```
Optimal hyperParameter: {'n_estimators': 500, 'max_depth': 8}

Maximum accuracy: 80.58571428571429
```

In [212]:

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

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

plot_confusion_matrix(y_test_tw, predicted_y)
```

```
Total number of data points : 15000
```



In [213]:

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 8
params['n_estimators'] = 500

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

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

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

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

```
[0]     train-logloss:0.683366  valid-logloss:0.684147
Multiple eval metrics have been passed: 'valid-logloss' will be used for
early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[1]     train-logloss:0.674006  valid-logloss:0.675557
[2]     train-logloss:0.665035  valid-logloss:0.667271
[3]     train-logloss:0.656335  valid-logloss:0.659339
[4]     train-logloss:0.647864  valid-logloss:0.651647
[5]     train-logloss:0.639855  valid-logloss:0.644289
[6]     train-logloss:0.63206   valid-logloss:0.637237
[7]     train-logloss:0.624535  valid-logloss:0.630405
[8]     train-logloss:0.617272  valid-logloss:0.623829
[9]     train-logloss:0.610305  valid-logloss:0.617514
[10]    train-logloss:0.603509  valid-logloss:0.611409
[11]    train-logloss:0.596951  valid-logloss:0.605531
[12]    train-logloss:0.590507  valid-logloss:0.59979
[13]    train-logloss:0.58435   valid-logloss:0.594323
[14]    train-logloss:0.578289  valid-logloss:0.588908
```

Conclusion :

Detailed step by step procedure i followed to solve this case study.

- The data set contains 404,290 rows and 5 columns : qid1, qid2, question1, question2, is_duplicate from which 'is_duplicate' is a class lable which specify that the question 1 and question 2 is similar or not , based on binary class label , we can difinee this is a binary classification problem.
- Firstly we do preprocess the data, then did feature engineering to create new features which might help us to slove problem and created our dataframes based on feature engineering , then we merged dataframes and got out final matrix. after doing simple EDA on dataset we will try some Basic Feature Extraction (before cleaning) the datset like Frequency of qid1's ,word_Common and word_share, etc. and using this featured datset we will do some EDA on it so that we will able to rectify which features features are most helpful for classification.

- After doing basic Basic feature extractions we will try some Advanced Feature Extraction using NLP and Fuzzy Features as per the documentation links provided above but before doing Advanced Feature Extraction we will do Preprocessing of Text and then we will do Advanced Feature Extraction and try to visualise our Advanced Feature using EDA, PCA and word clouds.
- Then we Splitted the data randomly . We could also have done time based splitting, since the model could predict for future unseen data too. But, there was no timestamp column provided, so the only option we have to split it randomly.
- We have columns of two questions i.e question1 and question2 and we will vectorize that both col using tfidf weighted word-vectors so that we will able to apply models on it and after doing all these we will merge all the features i.e besic features + advance features + question1 tfidf w2v + and question 2 tfidf w2v. and Now after doing all of there we will apply models on it.
- In this case study we have used log-loss and confusion matrix as a performance matrix to get performence of the models
- Applied models like Logistic Regression ,linear svm and XgBoost on random model which Finding worst-case log-loss .
- later applied models on tfidf vectorizer with hyperparameter tuning in order to improve the model performance.
- we got log loss of 0.88 on Random/Dumb Model. This is the worst case log-loss. This will act as a base model and any model we design should have a log-loss lesser than this dumb model.
- i applied Logistic Refreesion with hyperparameter, Linear SVM, XGBoost on Random Model (TFIDF W2v) .
- Similarlly i applied TFIDF model also .
- Applied Logistic Regression with hyperparameter tuning on TFIDF Model , It gave a log-loss of 0.42, which is lower than Random Model.
- Applied Linear SVM with hyperparameter tuning on TFIDF model, It gave a log-loss of 0.43, which is lower than Random Model.
- Finally applied Xgboost with hyperparameter tuning on TFIDF model . It gave the log-loss of 0.34,which is lower than Random Model.
- Xgboost seems to perform well and hence can be used to Identify which questions asked on Quora are duplicates of questions that have already been asked.

In [2]:

```
from prettytable import PrettyTable
p = PrettyTable()
p.title = " Model Comparision "
p.field_names = [ 'Model Name', 'Vectorizer','Hyperparameter Tunning', 'Test Log Loss']
p.add_row(["Random/Dumb","TFIDF Weighted W2V","-","0.88"])
p.add_row(["Logistic Regression","TFIDF Weighted W2V","Done","0.46"])
p.add_row(["Linear SVM","TFIDF Weighted W2V","Done","0.48"])
p.add_row(["XGBoost","TFIDF Weighted W2V","Done","0.35"])
p.add_row(["\n","\n","\n","\n"])
p.add_row(["Random/Dumb","TFIDF","-","0.88"])
p.add_row(["Logistic Regression","TFIDF","Done","0.42"])
p.add_row(["Linear SVM","TFIDF","Done","0.43"])
p.add_row(["XGBoost","TFIDF","Done","0.34"])
print(p)
```

```
+---------------------+--------------------+------------------------+-----
----------+
|      Model Name     |     Vectorizer     | Hyperparameter Tunning | Test
Log Loss |
+---------------------+--------------------+------------------------+-----
----------+
|     Random/Dumb     | TFIDF Weighted W2V |           -            |
0.88     |
| Logistic Regression | TFIDF Weighted W2V |          Done          |
0.46     |
|      Linear SVM     | TFIDF Weighted W2V |          Done          |
0.48     |
|       XGBoost       | TFIDF Weighted W2V |          Done          |
0.35     |
|                     |                    |                        |
|
|                     |                    |                        |
|
|     Random/Dumb     |       TFIDF        |           -            |
0.88     |
| Logistic Regression |       TFIDF        |          Done          |
0.42     |
|      Linear SVM     |       TFIDF        |          Done          |
0.43     |
|       XGBoost       |       TFIDF        |          Done          |
0.34     |
+---------------------+--------------------+------------------------+-----
----------+
```

Sign Off RAMESH BATTU (https://www.linkedin.com/in/rameshbattuai/)