Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> 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 ha
ppen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a great g
eologist?","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):

- Primary metric-Log-loss : This is because we just do not need the class label, we require the probability of the similiarity https://www.kaggle.com/wiki/LogarithmicLoss (https://www.kaggle.com/wiki/LogarithmicLoss)
- Secondary metric-Binary Confusion Matrix : This gives an indication about the false positives & false negatives as they are very importatnt in this case study.

2.3 Train and Test Construction

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

3. Exploratory Data Analysis and featurizations

Importing all the modules

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
from chart_studio.plotly import plotly
import plotly.graph_objs as go
#offline.init_notebook_mode()
from collections import Counter
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.tools as tls
import gc
import distance
from bs4 import BeautifulSoup
```

3.1 Reading data and basic stats

In [3]:

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

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

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

In [4]:

```
df.head()
```

Out[4]:

	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 [5]:

```
df.info()
```

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

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

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

3.2.1 Distribution of data points among output classes

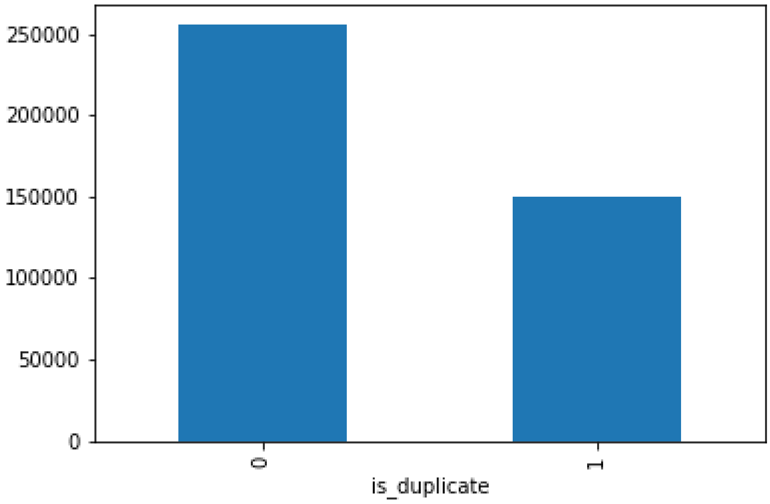
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [6]:

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

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x189518e3c88>
```



In [7]:

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

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

In [8]:

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

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

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

3.2.2 Number of unique questions

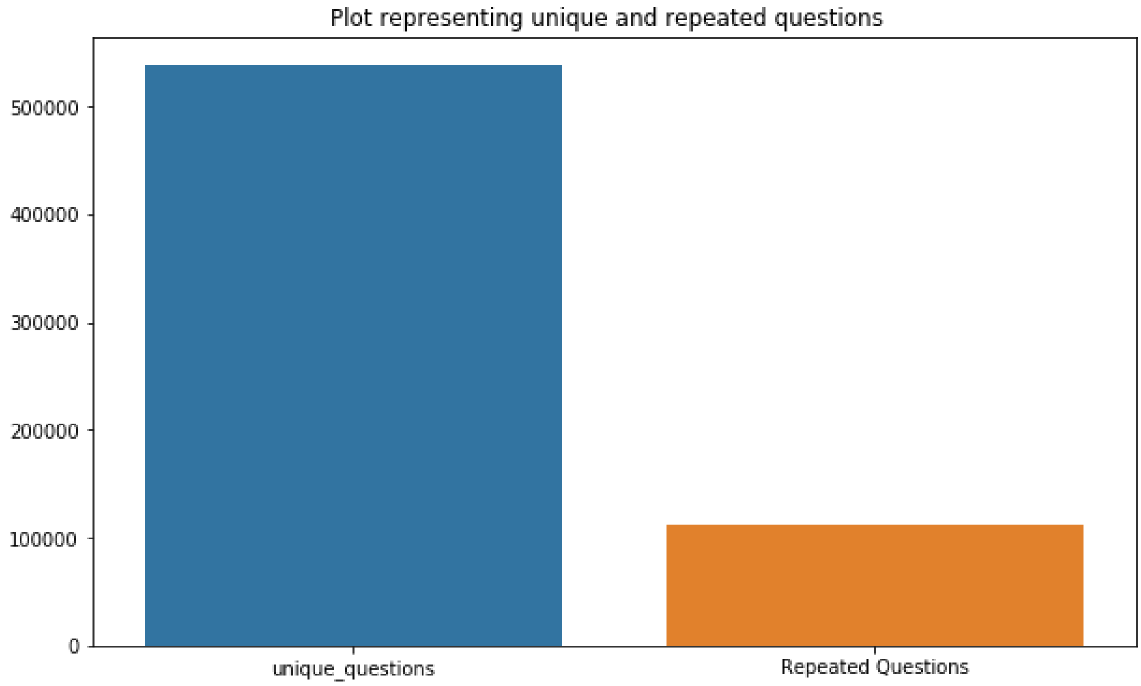
In [9]:

```
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
s_morethan_onetime,qs_morethan_onetime/unique_qs*100))

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.7795
3945937505%)

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

In [10]:

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

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



3.2.3 Checking for Duplicate rows in the dataset

In [11]:

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

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

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

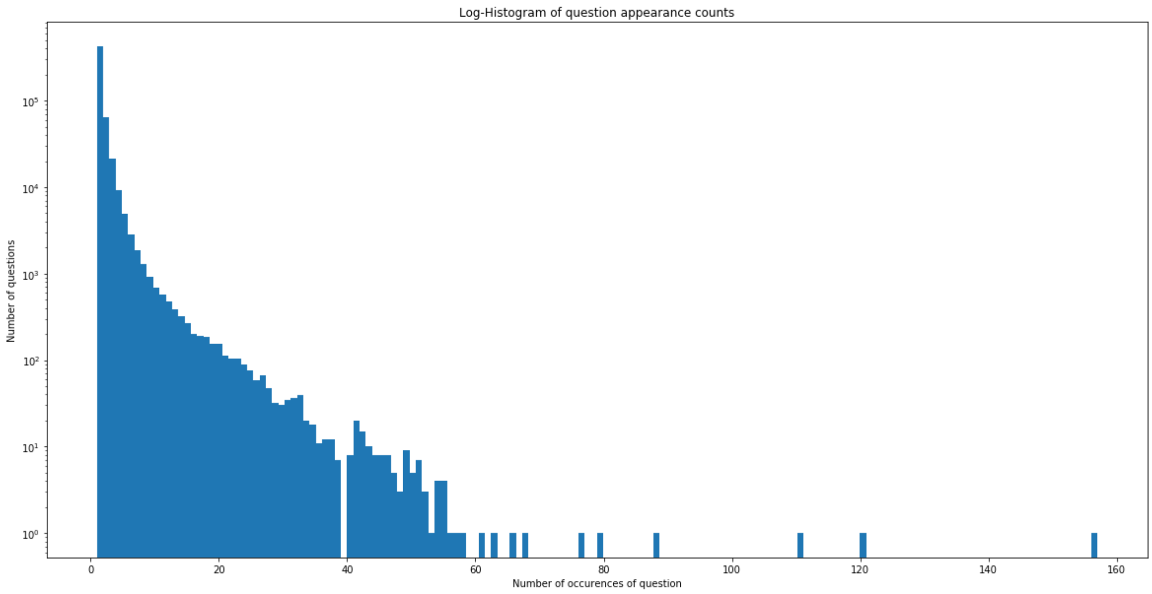
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [12]:

```
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
lue_counts())))
```

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



3.2.5 Checking for NULL values

In [13]:

```
#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 and 1 row null value in question1

In [14]:

```
# 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** = String Length of q1
- **q2len** = String 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 [15]:

```
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[15]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	5
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	8
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	5
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	6
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	3

In [16]:

```
df.shape #11 features have been added
```

Out[16]:

```
(404290, 17)
```

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [17]:

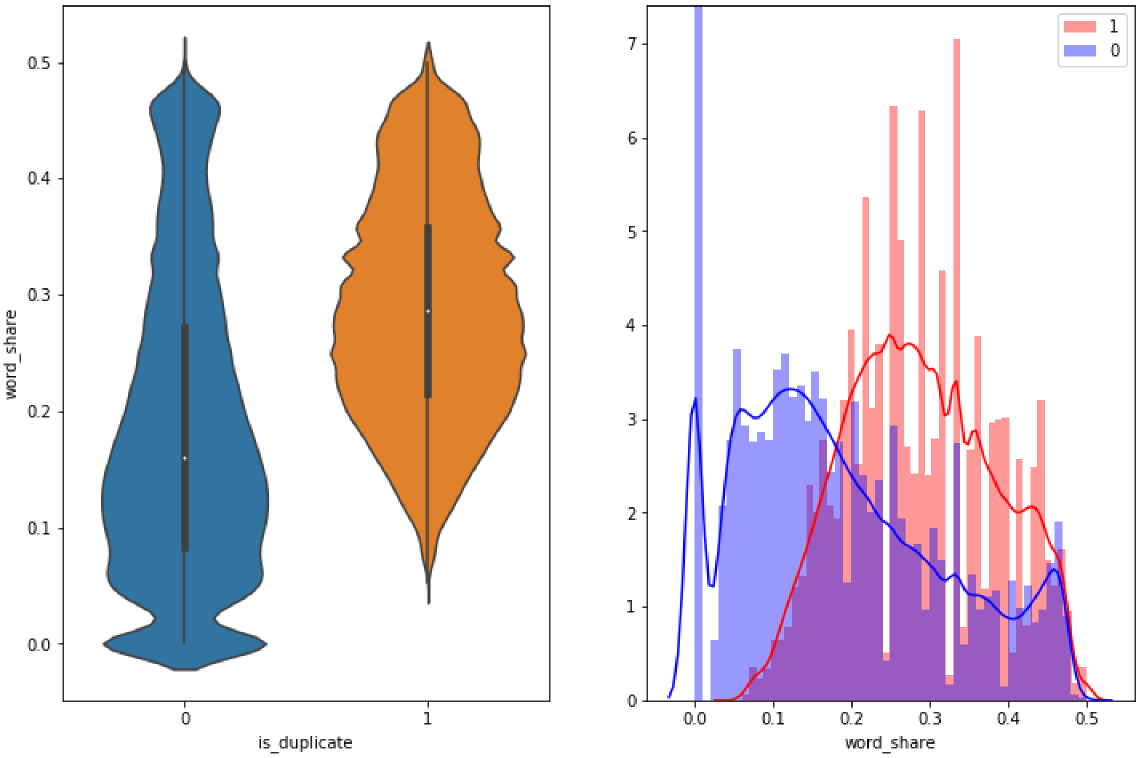
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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

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

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

3.3.1.1 Feature: word_share

In [18]:

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

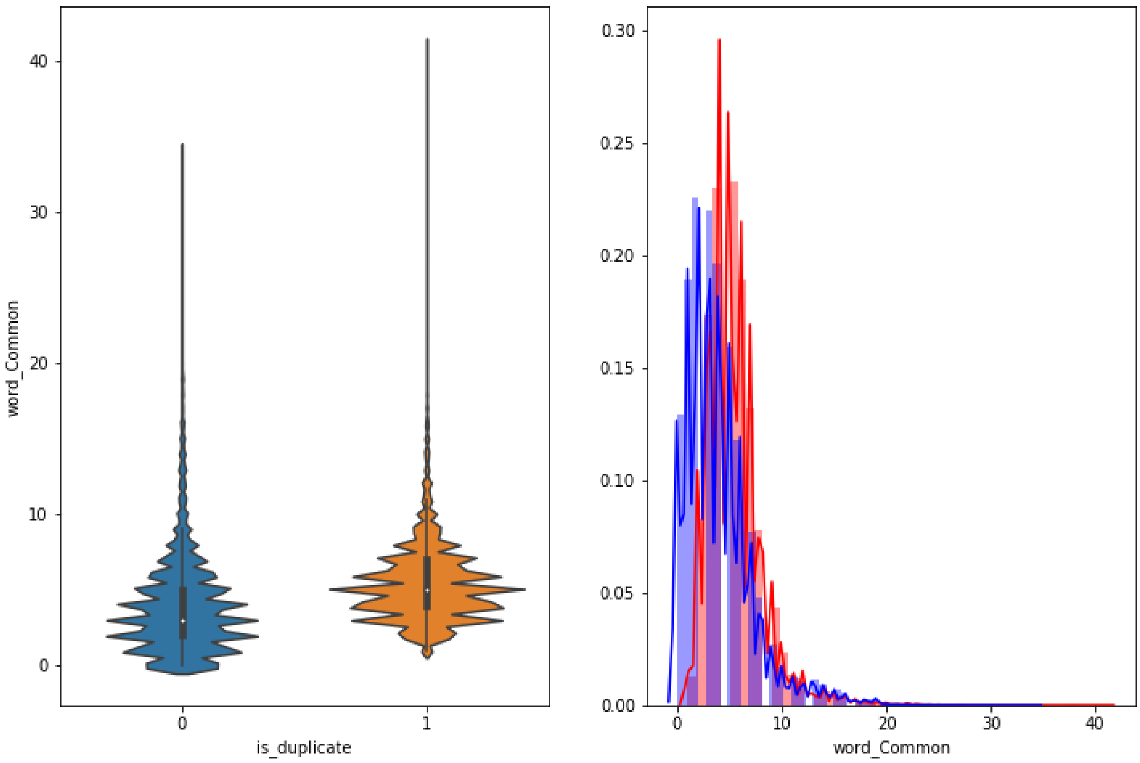


- From the dist. plot, the distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- From the box plot, the average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [19]:

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



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

1.2.1 : Importing few modules

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc
import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

In [21]:

```
df.head(2)
```

Out[21]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2l
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

3.4 Preprocessing of Text

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

In [23]:

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

STOP_WORDS = stopwords.words("english")


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


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

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


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


    return x
```

- 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 length 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 length 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 length 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 length 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 length 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 length of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

- **last_word_eq** : Check if Last 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 [24]:

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

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

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

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

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

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
```

```
        return len(strs[0]) / (min(len(a), len(b)) + 1)

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

    print("token features...")

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"
], x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the t
okens alphabetically, and
    # then joining them back into a string We then compare the transformed strings with
a simple ratio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question
1"], x["question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["que
stion2"]), axis=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"],
x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["questi
on1"], x["question2"]), axis=1)
    return df
```

In [25]:

```
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[25]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min
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
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

2 rows × 21 columns

In [29]:

```
df.columns
```

Out[29]:

```
Index(['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_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [30]:

```
df.shape
```

Out[30]:

```
(404290, 21)
```

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 [32]:

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

# Converting 2d array of q1 and q2 and flatten the array: like {{1,2},{3,4}} to {1,2,3,
4}
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).flatten()
#https://stackoverflow.com/questions/25116595/understanding-numpys-dstack-function

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

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

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

In [0]:

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

textp_w = open(path.join(d, 'train_p.txt')).read()
textn_w = open(path.join(d, 'train_n.txt')).read()
stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

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

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193130
```

Word Clouds generated from duplicate pair question's text

In [0]:

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

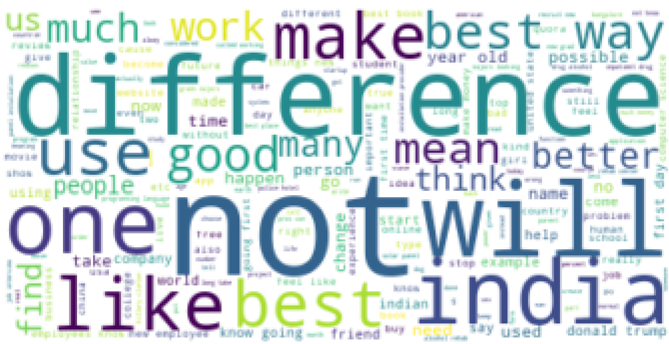
```
Word Cloud for Duplicate Question pairs
```

Word Clouds generated from non duplicate pair question's text

In [0]:

```
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()
```

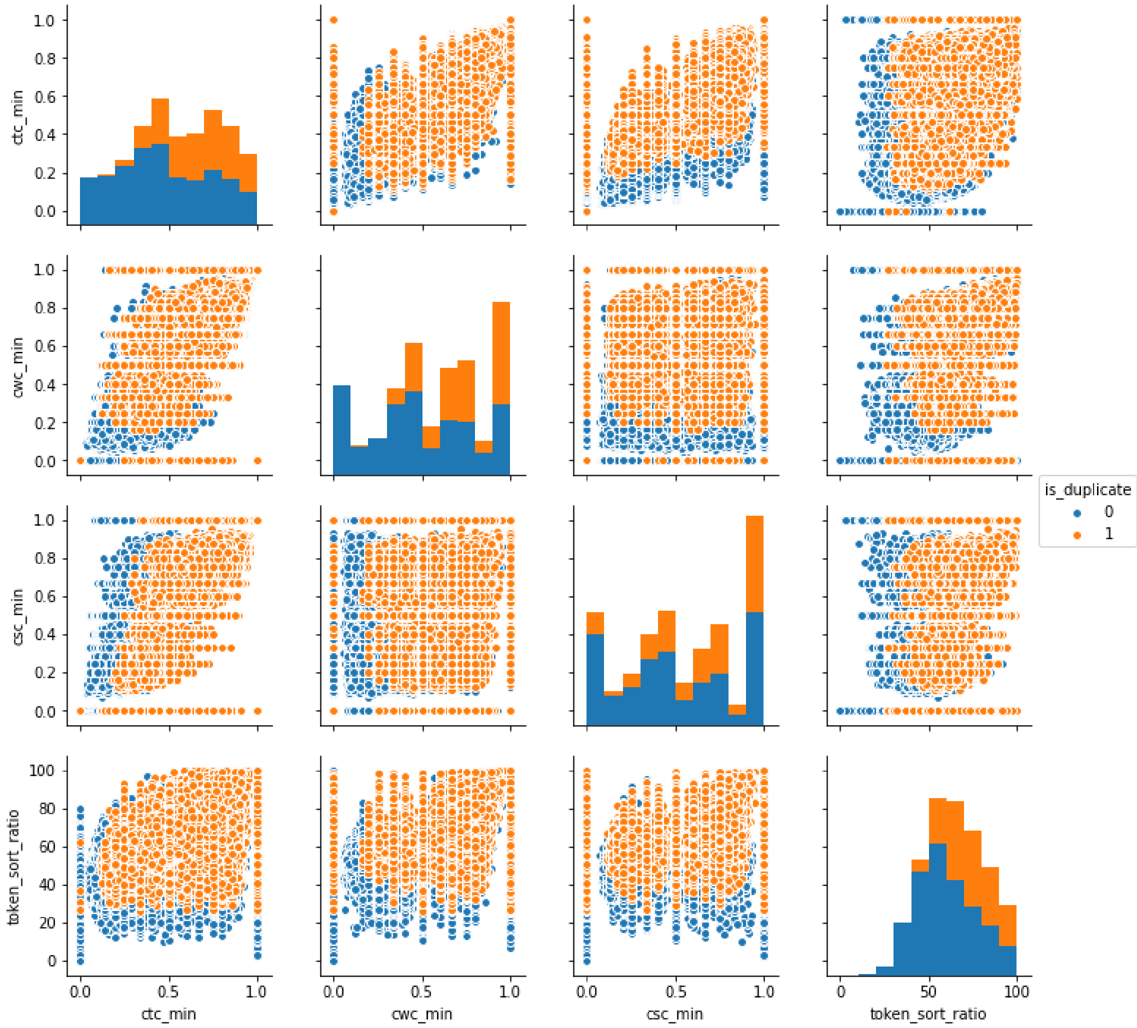
```
Word Cloud for non-Duplicate Question pairs:
```



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

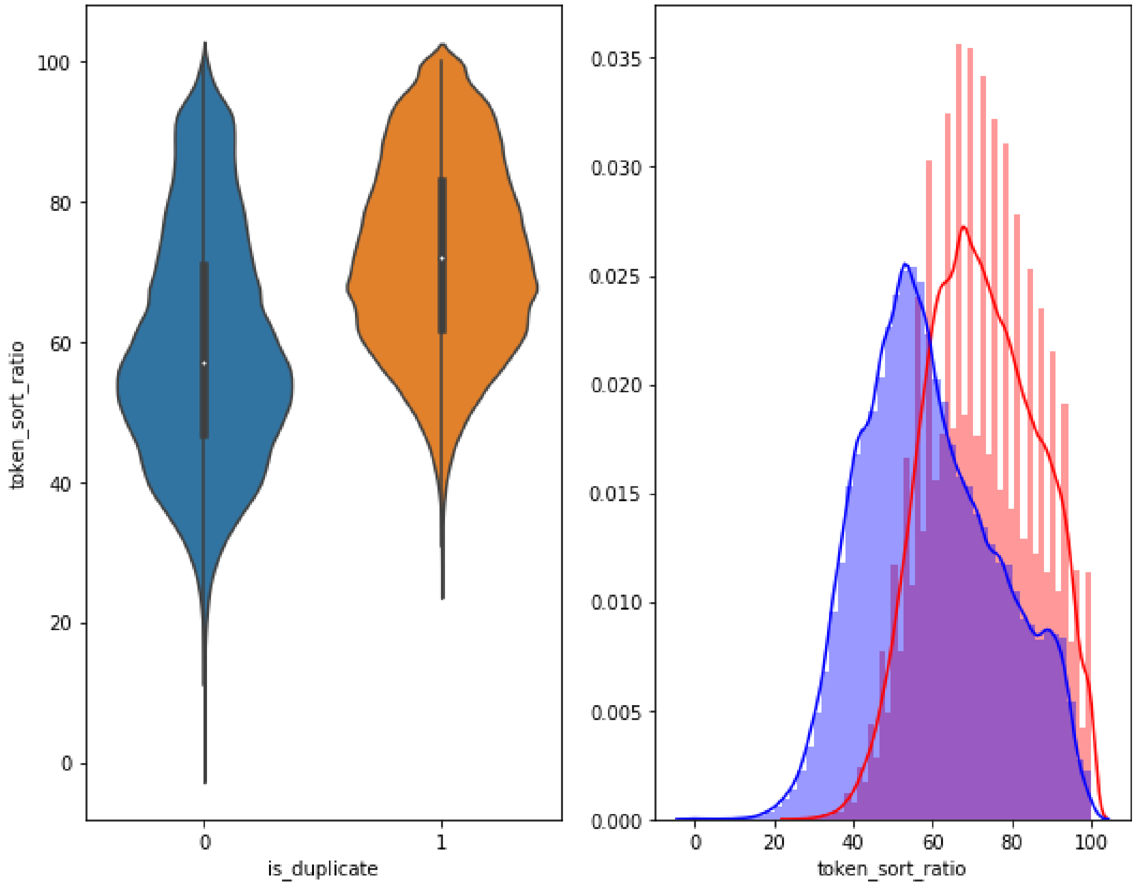
In [0]:

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



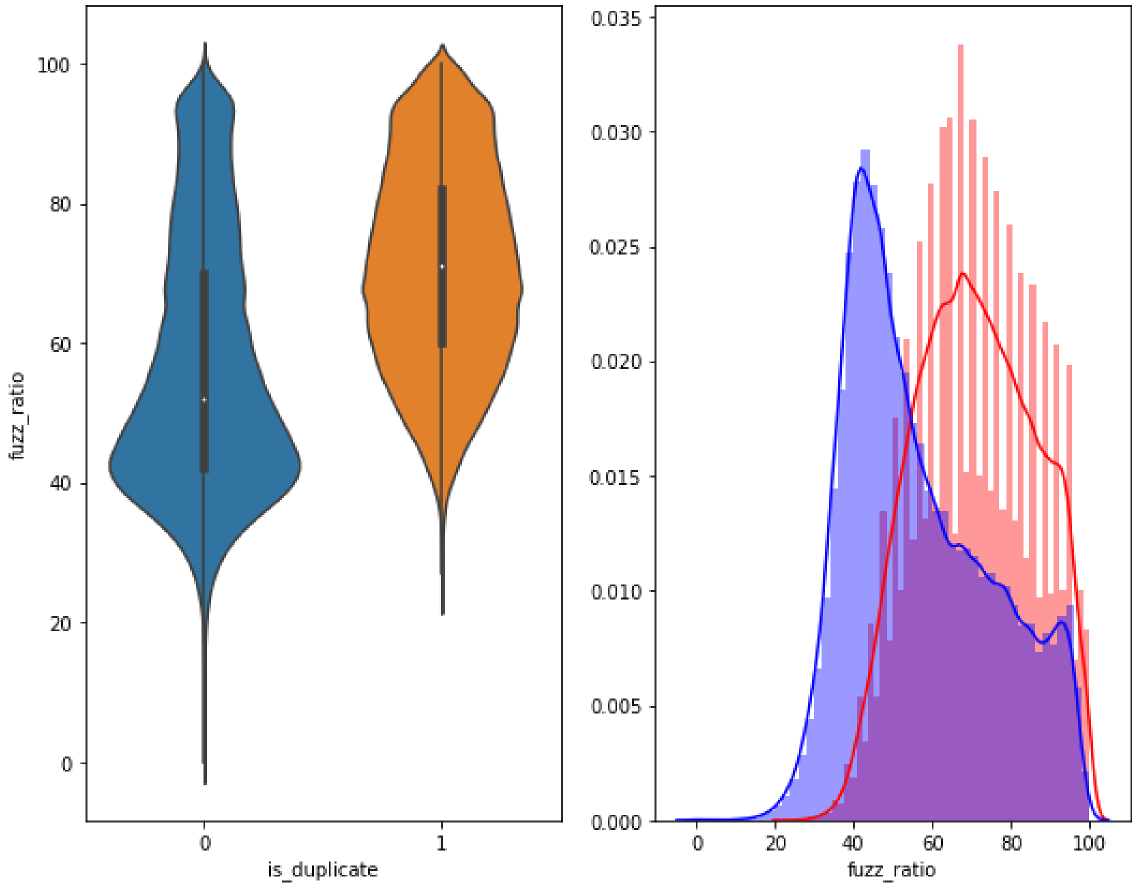
In [0]:

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

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

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



In [0]:

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



3.5.2 Visualization

In [0]:

```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the
 data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['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_partial_ratio' ,
'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

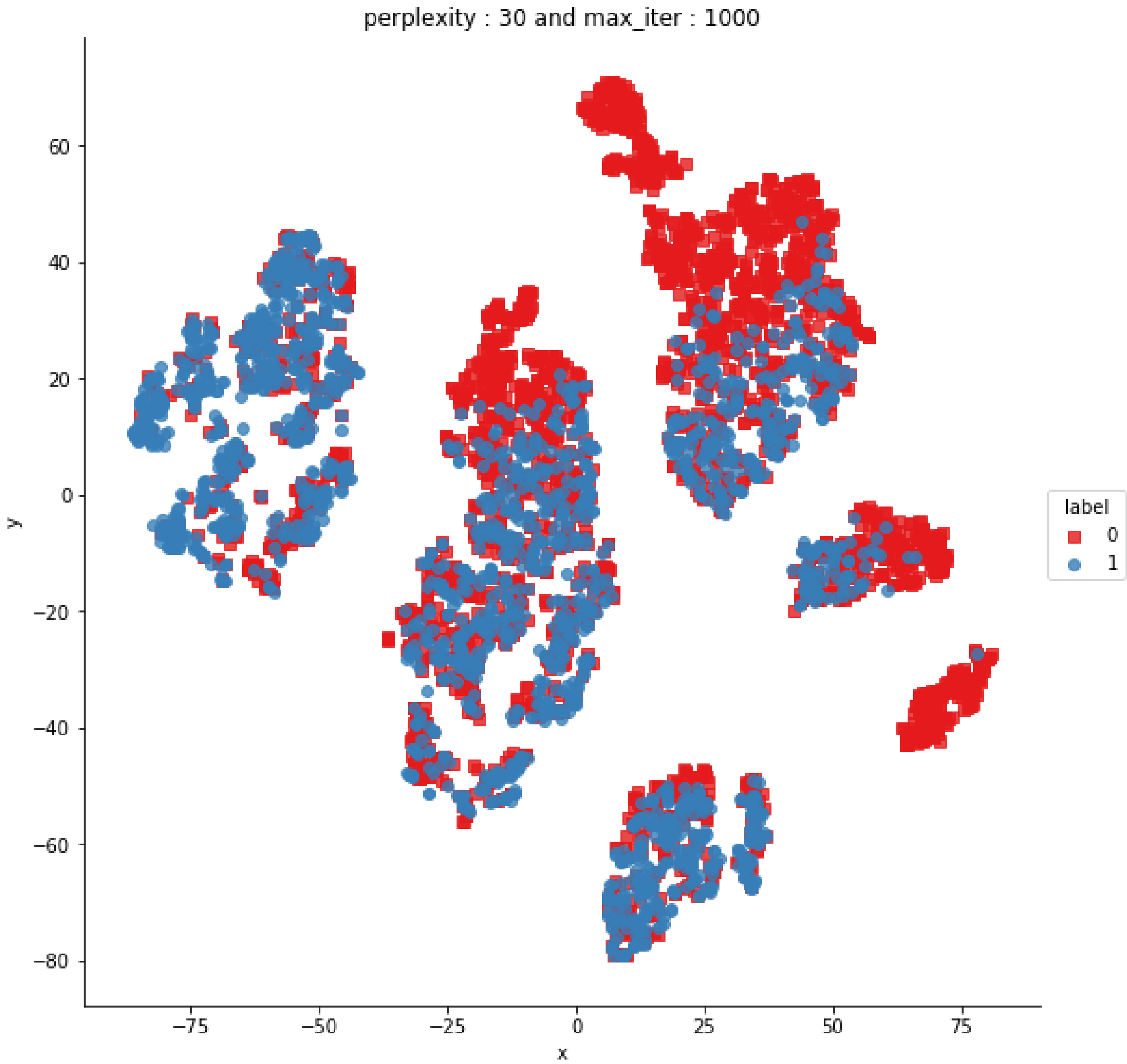
In [0]:

```
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 0.011s...
[t-SNE] Computed neighbors for 5000 samples in 0.912s...
[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 0.433s
[t-SNE] Iteration 50: error = 80.9244080, gradient norm = 0.0428133 (50 it
erations in 13.099s)
[t-SNE] Iteration 100: error = 70.3858795, gradient norm = 0.0100968 (50 i
terations in 9.067s)
[t-SNE] Iteration 150: error = 68.6138382, gradient norm = 0.0058392 (50 i
terations in 9.602s)
[t-SNE] Iteration 200: error = 67.7700119, gradient norm = 0.0036596 (50 i
terations in 9.121s)
[t-SNE] Iteration 250: error = 67.2725067, gradient norm = 0.0034962 (50 i
terations in 11.305s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.272
507
[t-SNE] Iteration 300: error = 1.7737305, gradient norm = 0.0011918 (50 it
erations in 8.289s)
[t-SNE] Iteration 350: error = 1.3720417, gradient norm = 0.0004822 (50 it
erations in 10.526s)
[t-SNE] Iteration 400: error = 1.2039998, gradient norm = 0.0002768 (50 it
erations in 9.600s)
[t-SNE] Iteration 450: error = 1.1133438, gradient norm = 0.0001881 (50 it
erations in 11.827s)
[t-SNE] Iteration 500: error = 1.0579143, gradient norm = 0.0001434 (50 it
erations in 8.941s)
[t-SNE] Iteration 550: error = 1.0221983, gradient norm = 0.0001164 (50 it
erations in 11.092s)
[t-SNE] Iteration 600: error = 0.9987167, gradient norm = 0.0001039 (50 it
erations in 11.467s)
[t-SNE] Iteration 650: error = 0.9831534, gradient norm = 0.0000938 (50 it
erations in 11.799s)
[t-SNE] Iteration 700: error = 0.9722011, gradient norm = 0.0000858 (50 it
erations in 12.028s)
[t-SNE] Iteration 750: error = 0.9643636, gradient norm = 0.0000799 (50 it
erations in 12.120s)
[t-SNE] Iteration 800: error = 0.9584482, gradient norm = 0.0000785 (50 it
erations in 11.867s)
[t-SNE] Iteration 850: error = 0.9538348, gradient norm = 0.0000739 (50 it
erations in 11.461s)
[t-SNE] Iteration 900: error = 0.9496906, gradient norm = 0.0000712 (50 it
erations in 11.023s)
[t-SNE] Iteration 950: error = 0.9463405, gradient norm = 0.0000673 (50 it
erations in 11.755s)
[t-SNE] Iteration 1000: error = 0.9432716, gradient norm = 0.0000662 (50 i
terations in 11.493s)
[t-SNE] Error after 1000 iterations: 0.943272
```

In [0]:

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



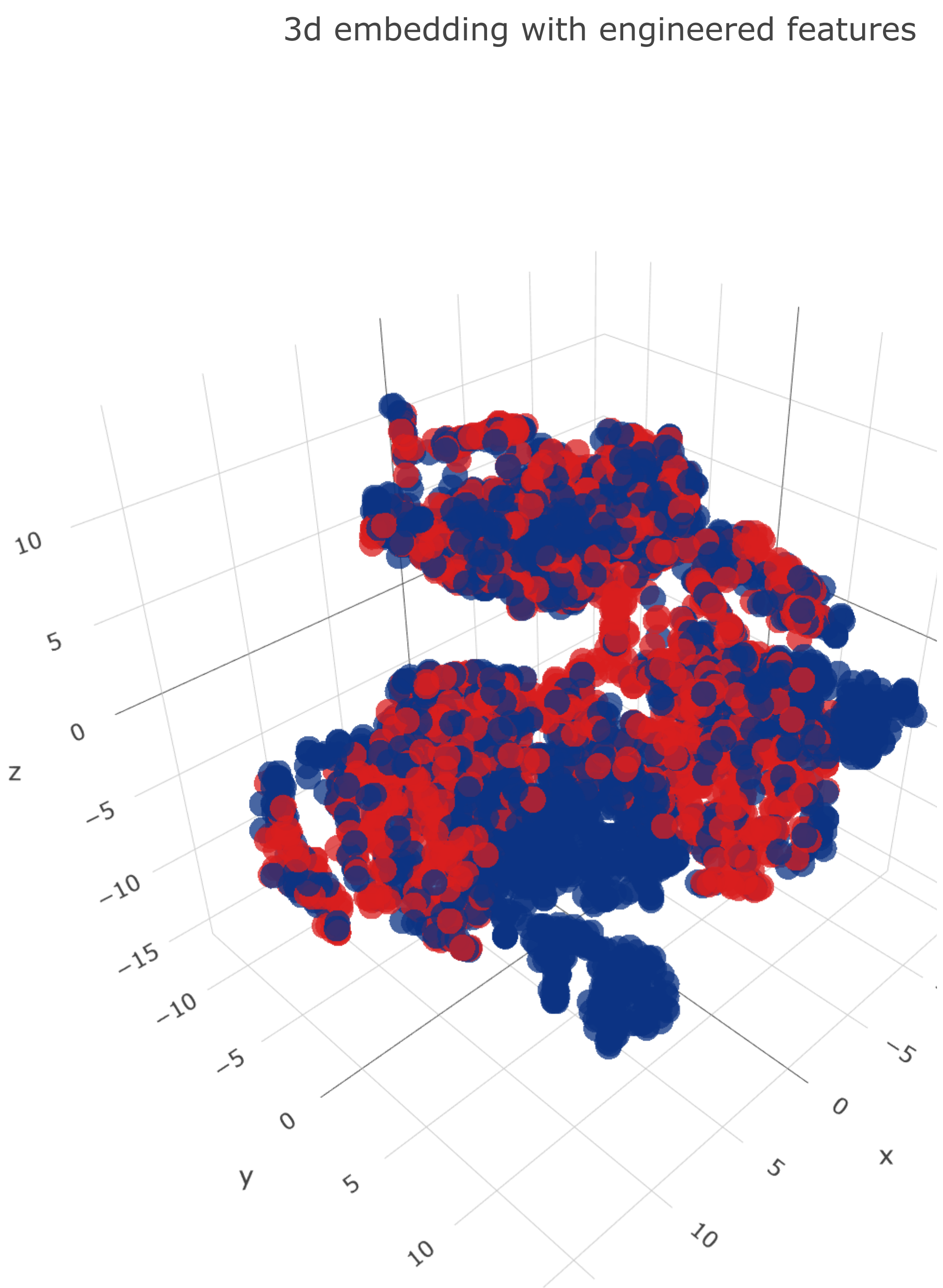
In [0]:

```
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 0.010s...
[t-SNE] Computed neighbors for 5000 samples in 0.935s...
[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 0.363s
[t-SNE] Iteration 50: error = 77.7944183, gradient norm = 0.1014017 (50 it
erations in 34.931s)
[t-SNE] Iteration 100: error = 69.2682266, gradient norm = 0.0248657 (50 i
terations in 15.147s)
[t-SNE] Iteration 150: error = 67.7877655, gradient norm = 0.0150941 (50 i
terations in 13.761s)
[t-SNE] Iteration 200: error = 67.1991119, gradient norm = 0.0126559 (50 i
terations in 13.425s)
[t-SNE] Iteration 250: error = 66.8560715, gradient norm = 0.0074975 (50 i
terations in 12.904s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.856
071
[t-SNE] Iteration 300: error = 1.2356015, gradient norm = 0.0007033 (50 it
erations in 13.302s)
[t-SNE] Iteration 350: error = 0.9948602, gradient norm = 0.0001997 (50 it
erations in 18.898s)
[t-SNE] Iteration 400: error = 0.9168936, gradient norm = 0.0001430 (50 it
erations in 13.397s)
[t-SNE] Iteration 450: error = 0.8863022, gradient norm = 0.0000975 (50 it
erations in 16.379s)
[t-SNE] Iteration 500: error = 0.8681002, gradient norm = 0.0000854 (50 it
erations in 17.791s)
[t-SNE] Iteration 550: error = 0.8564141, gradient norm = 0.0000694 (50 it
erations in 17.060s)
[t-SNE] Iteration 600: error = 0.8470711, gradient norm = 0.0000640 (50 it
erations in 15.454s)
[t-SNE] Iteration 650: error = 0.8389117, gradient norm = 0.0000561 (50 it
erations in 17.562s)
[t-SNE] Iteration 700: error = 0.8325295, gradient norm = 0.0000529 (50 it
erations in 13.443s)
[t-SNE] Iteration 750: error = 0.8268463, gradient norm = 0.0000528 (50 it
erations in 17.981s)
[t-SNE] Iteration 800: error = 0.8219477, gradient norm = 0.0000477 (50 it
erations in 17.448s)
[t-SNE] Iteration 850: error = 0.8180174, gradient norm = 0.0000490 (50 it
erations in 18.376s)
[t-SNE] Iteration 900: error = 0.8150476, gradient norm = 0.0000456 (50 it
erations in 17.778s)
[t-SNE] Iteration 950: error = 0.8122067, gradient norm = 0.0000472 (50 it
erations in 16.983s)
[t-SNE] Iteration 1000: error = 0.8095787, gradient norm = 0.0000489 (50 i
terations in 18.581s)
[t-SNE] Error after 1000 iterations: 0.809579
```

In [0]:

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

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



4. Merging data frames, splitting the data and scaling the numerical extracted features

4.1 Combining dataframes of basic & advanced features

In [3]:

```
df_basic=pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
df_basic.head(2)
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2l
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

In [4]:

```
df_basic.columns
```

Out[4]:

```
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')
```

In [5]:

```
df_advanced=pd.read_csv("nlp_features_train.csv",encoding='latin-1')
df_advanced.head(2)
```

Out[5]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	(
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	(
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	(

2 rows × 21 columns

In [6]:

```
df_advanced.columns
```

Out[6]:

```
Index(['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_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [7]:

```
df1=df_basic.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [8]:

```
df1.head()
```

Out[8]:

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

In [9]:

```
df1.shape
```

Out[9]:

(404290, 12)

In [10]:

```
data=df_advanced.merge(df1, on='id',how='left')
```

In [11]:

```
data.head(2)
```

Out[11]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	(
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	(
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	(

2 rows × 32 columns

In [12]:

```
data.shape
```

Out[12]:

```
(404290, 32)
```

In [49]:

```
export_csv = data.to_csv (r'D:\PGS\Applied AI course\E-Notes\Module_6-Real World Case s
tudies\Case Study 1- Quora question Pair Similarity Problem\my_data.csv', index = None,
header=True)
```

In [13]:

```
data = pd.read_csv('my_data.csv', nrows=100000)
data.head(2)
```

Out[13]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	(
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	(
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	(

2 rows × 32 columns

4.2 Splitting the data

In [14]:

```
Y=data['is_duplicate']
X=data.drop(['qid1','qid2','is_duplicate'],axis=1)
print(X.shape)
print(Y.shape)
```

```
(100000, 29)
(100000,)
```

In [15]:

```
X.head(2)
```

Out[15]:

	id	question1	question2	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_
0	0	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0.999980	0.833319	0.999983	0.999983	0.916659	0.785
1	1	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0.799984	0.399996	0.749981	0.599988	0.699993	0.466

2 rows × 29 columns

In [16]:

```
# Random train test split( 70:30)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, stratify=Y)
```

In [17]:

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

```
Number of data points in train data : (70000, 29)
Number of data points in test data : (30000, 29)
```

In [18]:

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

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

4.3 Normalizing extracted features(Basic & Advanced)

4.3.1 Advanced features

In [81]:

```
def scaler(train_column,test_column):
    from sklearn.preprocessing import Normalizer
    normalizer = Normalizer()
    normalizer.fit(train_column.values.reshape(-1,1))

    train_norm = normalizer.transform(train_column.values.reshape(-1,1))
    test_norm = normalizer.transform(test_column.values.reshape(-1,1))

    return train_norm,test_norm
```

In [51]:

```
X_train_cwc_min_norm,X_test_cwc_min_norm = scaler(X_train['cwc_min'],X_test['cwc_min'])
print("After vectorizations")
print(X_train_cwc_min_norm.shape, y_train.shape)
print(X_test_cwc_min_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(70000, 1) (70000,)
(30000, 1) (30000,)
==========================================================================
==========================
```

In [52]:

```
X_train_cwc_max_norm,X_test_cwc_max_norm = scaler(X_train['cwc_max'],X_test['cwc_max'])

X_train_csc_min_norm,X_test_csc_min_norm = scaler(X_train['csc_min'],X_test['csc_min'])

X_train_csc_max_norm,X_test_csc_max_norm = scaler(X_train['csc_max'],X_test['csc_max'])

X_train_ctc_min_norm,X_test_ctc_min_norm = scaler(X_train['ctc_min'],X_test['ctc_min'])

X_train_ctc_max_norm,X_test_ctc_max_norm = scaler(X_train['ctc_max'],X_test['ctc_max'])

X_train_last_word_eq_norm,X_test_last_word_eq_norm = scaler(X_train['last_word_eq'],X_t
est['last_word_eq'])

X_train_first_word_eq_norm,X_test_first_word_eq_norm = scaler(X_train['first_word_eq'],
X_test['first_word_eq'])

X_train_abs_len_diff_norm,X_test_abs_len_diff_norm = scaler(X_train['abs_len_diff'],X_t
est['abs_len_diff'])

X_train_mean_len_norm,X_test_mean_len_norm = scaler(X_train['mean_len'],X_test['mean_le
n'])

X_train_token_set_ratio_norm,X_test_token_set_ratio_norm = scaler(X_train['token_set_ra
tio'],X_test['token_set_ratio'])

X_train_token_sort_ratio_norm,X_test_token_sort_ratio_norm = scaler(X_train['token_sort
_ratio'],X_test['token_sort_ratio'])

X_train_fuzz_ratio_norm,X_test_fuzz_ratio_norm = scaler(X_train['fuzz_ratio'],X_test['f
uzz_ratio'])

X_train_fuzz_partial_ratio_norm,X_test_fuzz_partial_ratio_norm = scaler(X_train['fuzz_p
artial_ratio'],X_test['fuzz_partial_ratio'])

X_train_longest_substr_ratio_norm,X_test_longest_substr_ratio_norm = scaler(X_train['lo
ngest_substr_ratio'],X_test['longest_substr_ratio'])
```

4.3.2 Basic features

In [53]:

```
X_train_freq_qid1_norm,X_test_freq_qid1_norm = scaler(X_train['freq_qid1'],X_test['freq
_qid1'])
print("After vectorizations")
print(X_train_freq_qid1_norm.shape, y_train.shape)
print(X_test_freq_qid1_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(70000, 1) (70000,)
(30000, 1) (30000,)
==========================================================================
==========================
```

In [54]:

```
X_train.columns
```

Out[54]:

```
Index(['id', 'question1', 'question2', '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_partial_ratio', 'longest_substr_ratio', 'freq_q
id1',
       'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q
2'],
      dtype='object')
```

In [55]:

```
X_train_freq_qid2_norm,X_test_freq_qid2_norm = scaler(X_train['freq_qid2'],X_test['freq
_qid2'])

X_train_q1len_norm,X_test_q1len_norm = scaler(X_train['q1len'],X_test['q1len'])

X_train_q2len_norm,X_test_q2len_norm = scaler(X_train['q2len'],X_test['q2len'])

X_train_q1_n_words_norm,X_test_q1_n_words_norm = scaler(X_train['q1_n_words'],X_test['q
1_n_words'])

X_train_q2_n_words_norm,X_test_q2_n_words_norm = scaler(X_train['q2_n_words'],X_test['q
2_n_words'])

X_train_word_Common_norm,X_test_word_Common_norm = scaler(X_train['word_Common'],X_test
['word_Common'])

X_train_word_Total_norm,X_test_word_Total_norm = scaler(X_train['word_Total'],X_test['w
ord_Total'])

X_train_word_share_norm,X_test_word_share_norm = scaler(X_train['word_share'],X_test['w
ord_share'])

X_train_freq_q1addq2_norm,X_test_freq_q1addq2_norm = scaler(X_train['freq_q1+q2'],X_tes
t['freq_q1+q2'])

X_train_freq_q1subq2_norm,X_test_freq_q1subq2_norm = scaler(X_train['freq_q1-q2'],X_tes
t['freq_q1-q2'])
```

5. Featurizing text data using TFIDF & building models

5.1 TFIDF vectorization

TFIDF on 'question1' column

In [77]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer1 = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=3000)
vectorizer1.fit(X_train['question1'].values.astype('U'))

X_train_q1_tfidf = vectorizer1.transform(X_train['question1'].values.astype('U'))
X_test_q1_tfidf = vectorizer1.transform(X_test['question1'].values.astype('U'))
```

In [78]:

```
f1=vectorizer1.get_feature_names()
print("After vectorization")
print(X_train_q1_tfidf.shape, y_train.shape)
print(X_test_q1_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(70000, 3000) (70000,)
(30000, 3000) (30000,)
==========================================================================
==========================
```

TFIDF on 'question2' column

In [79]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer2 = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=3000)
vectorizer2.fit(X_train['question2'].values.astype('U'))

X_train_q2_tfidf = vectorizer2.transform(X_train['question2'].values.astype('U'))
X_test_q2_tfidf = vectorizer2.transform(X_test['question2'].values.astype('U'))
```

In [80]:

```
print("After vectorization")
print(X_train_q2_tfidf.shape, y_train.shape)
print(X_test_q2_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(70000, 3000) (70000,)
(30000, 3000) (30000,)
==========================================================================
==========================
```

5.2 Stacking TFIDF+ Basic + Advanced features

In [82]:

```
#for train data

from scipy.sparse import coo_matrix,hstack

X_tr_tfidf = hstack((X_train_q1_tfidf,X_train_q2_tfidf,X_train_cwc_min_norm,X_train_cwc
_max_norm,X_train_csc_min_norm,X_train_csc_max_norm,
                    X_train_ctc_min_norm,X_train_ctc_max_norm,X_train_last_word_eq_norm
,X_train_first_word_eq_norm,
                    X_train_abs_len_diff_norm,X_train_mean_len_norm,X_train_token_set_r
atio_norm,
                    X_train_token_sort_ratio_norm,X_train_fuzz_ratio_norm,X_train_fuzz_
partial_ratio_norm,
                    X_train_longest_substr_ratio_norm,X_train_freq_qid1_norm,X_train_fr
eq_qid2_norm,X_train_q1len_norm,X_train_q2len_norm,
                    X_train_q1_n_words_norm,X_train_q2_n_words_norm,X_train_word_Common
_norm,X_train_word_Total_norm,
                    X_train_word_share_norm,X_train_freq_q1addq2_norm,X_train_freq_q1su
bq2_norm,)).tocsr()
```

In [83]:

```
#for test data

X_test_tfidf = hstack((X_test_q1_tfidf,X_test_q2_tfidf,X_test_cwc_min_norm,X_test_cwc_m
ax_norm,X_test_csc_min_norm,X_test_csc_max_norm,
                    X_test_ctc_min_norm,X_test_ctc_max_norm,X_test_last_word_eq_norm,X_
test_first_word_eq_norm,
                    X_test_abs_len_diff_norm,X_test_mean_len_norm,X_test_token_set_rati
o_norm,
                    X_test_token_sort_ratio_norm,X_test_fuzz_ratio_norm,X_test_fuzz_par
tial_ratio_norm,
                    X_test_longest_substr_ratio_norm,X_test_freq_qid1_norm,X_test_freq_
qid2_norm,X_test_q1len_norm,X_test_q2len_norm,
                    X_test_q1_n_words_norm,X_test_q2_n_words_norm,X_test_word_Common_no
rm,X_test_word_Total_norm,
                    X_test_word_share_norm,X_test_freq_q1addq2_norm,X_test_freq_q1subq2
_norm,)).tocsr()
```

In [84]:

```
print("Final Data Matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final Data Matrix
(70000, 6026) (70000,)
(30000, 6026) (30000,)
```

In [85]:

```
# https://stackoverflow.com/questions/8955448/save-load-scipy-sparse-csr-matrix-in-port
able-data-format
from scipy import sparse
sparse.save_npz("X_tr_tfidf.npz", X_tr_tfidf)
sparse.save_npz("X_test_tfidf.npz", X_test_tfidf)
```

In [110]:

```
#https://www.geeksforgeeks.org/numpy-save/
np.save('y_train', y_train.values)
np.save('y_test', y_test.values)
```

5.3 Confusion matrix, Precision matrix & Recall matrix

In [2]:

```
# 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
ted class j

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

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

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

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

    labels = [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
bels)
    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
bels)
    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
bels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")
```

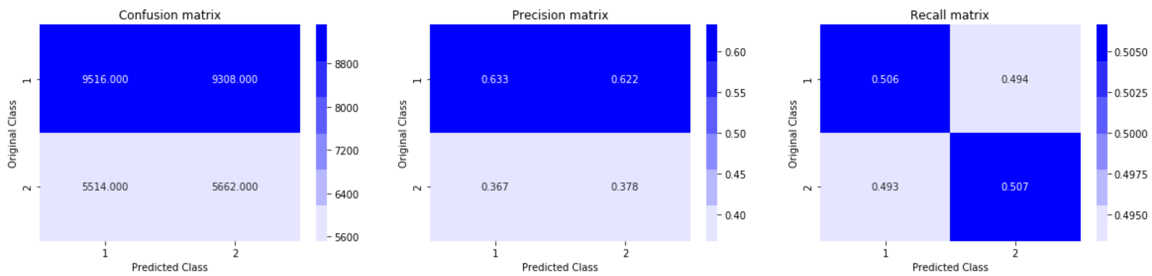
```
    plt.show()
```

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

In [88]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
from sklearn.metrics.classification import accuracy_score, log_loss

test_len = len(y_test)
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    #https://www.geeksforgeeks.org/numpy-random-rand-python/
    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
5))

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

```
Log loss on Test Data using Random Model 0.8803563782219028
```



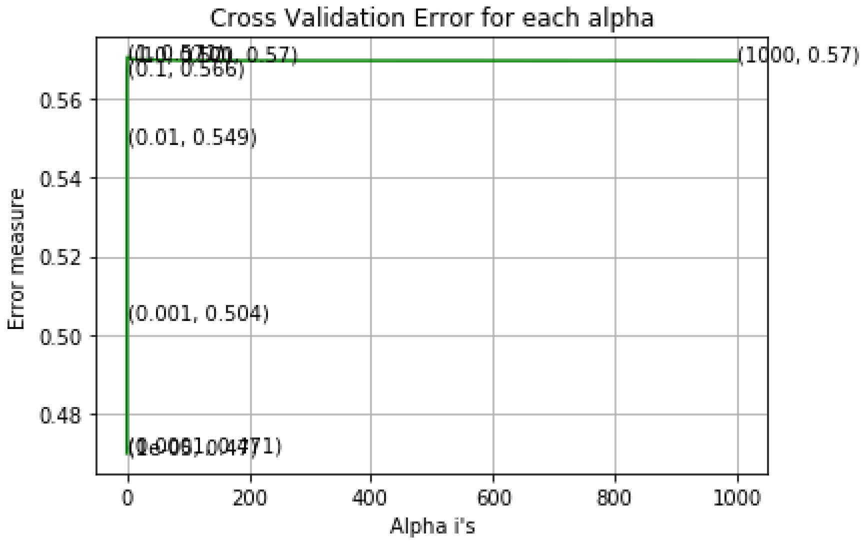
5.5 Logistic Regression with hyperparameter tuning

In [89]:

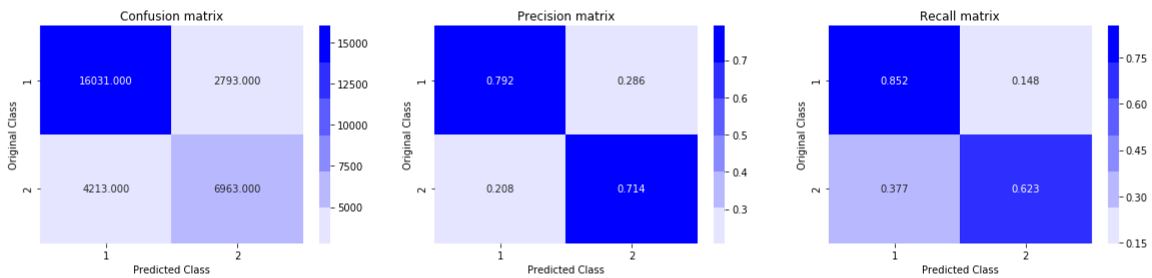
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

alpha = [10 ** x for x in range(-5, 4,)] # hyperparam for SGD classifier.

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

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

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



log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', class_weight='balanced',rand
om_state=42)
    clf.fit(X_tr_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    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
abels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(X_tr_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_tfidf)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
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: 0.4698712829204895
For values of alpha =  0.0001 The log loss is: 0.47067881405444
For values of alpha =  0.001 The log loss is: 0.5041594558899147
For values of alpha =  0.01 The log loss is: 0.5490340666106179
For values of alpha =  0.1 The log loss is: 0.5664228005052474
For values of alpha =  1 The log loss is: 0.570618098029543
For values of alpha =  10 The log loss is: 0.5700092824996266
For values of alpha =  100 The log loss is: 0.5696951887403758
For values of alpha =  1000 The log loss is: 0.5696694967125947
```




```
For values of best alpha =  1e-05 The train log loss is: 0.423380344999660
04
For values of best alpha =  1e-05 The test log loss is: 0.4698712829204895
Total number of data points : 30000
```



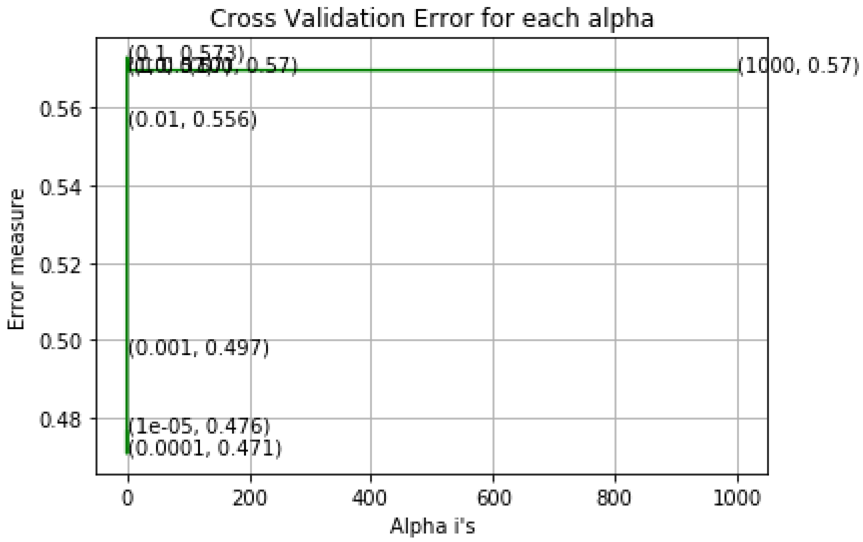
5.6 Linear SVM with hyperparameter tuning

In [90]:

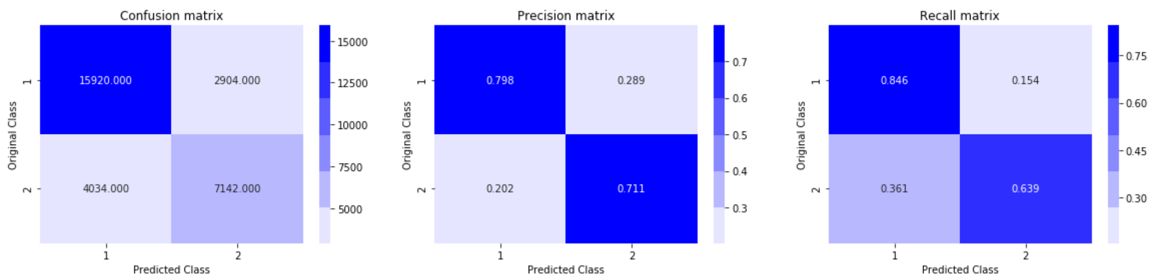
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

alpha = [10 ** x for x in range(-5, 4,)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', class_weight='balanced',ra
ndom_state=42)
    clf.fit(X_tr_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    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
abels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(X_tr_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_tfidf)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
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: 0.47642531219758116
For values of alpha =  0.0001 The log loss is: 0.47078579881124905
For values of alpha =  0.001 The log loss is: 0.49698778262848203
For values of alpha =  0.01 The log loss is: 0.5559012949642448
For values of alpha =  0.1 The log loss is: 0.5729869093875302
For values of alpha =  1 The log loss is: 0.5696802321772769
For values of alpha =  10 The log loss is: 0.5696783950237323
For values of alpha =  100 The log loss is: 0.5696786582960095
For values of alpha =  1000 The log loss is: 0.5696786584227757
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44005538166271
924
For values of best alpha =  0.0001 The test log loss is: 0.470785798811249
05
Total number of data points : 30000
```



5.7 XGBoost with hyperparameter tuning

In [3]:

```
from scipy import sparse
import numpy as np
X_tr_tfidf= sparse.load_npz("X_tr_tfidf.npz")
X_test_tfidf = sparse.load_npz("X_test_tfidf.npz")
y_train = np.load('y_train.npy')
y_test = np.load('y_test.npy')
```

In [9]:

```
#https://dask-ml.readthedocs.io/en/stable/modules/generated/dask_ml.xgboost.XGBClassifi
er.html
#https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import log_loss
from xgboost import XGBClassifier

xgb = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64],'max_depth': [4, 6, 8, 10, 12]}
clf1 = RandomizedSearchCV(xgb, parameters, cv=5, scoring='neg_log_loss',return_train_sc
ore=True,n_jobs=-1)
rs1 = clf1.fit(X_tr_tfidf, y_train)
```

In [10]:

```
df=pd.DataFrame(clf1.cv_results_)
df.head(2)
```

Out[10]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max
0	30.175921	0.419009	-0.504676	-0.492908	6
1	11.561633	0.385943	-0.536407	-0.528430	8

2 rows × 22 columns

In [21]:

```
df.to_csv(r'HYP.csv')
```

In [3]:

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

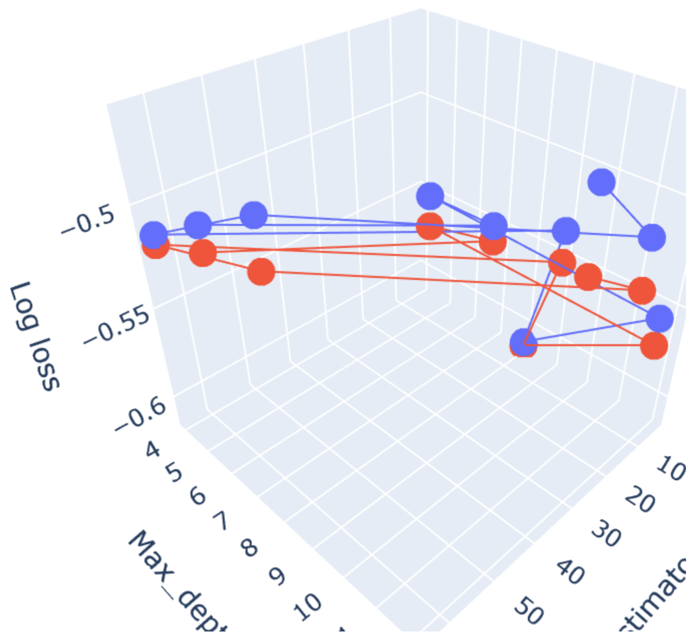
3D-Plot

In [4]:

```
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/requ
ire.js"></script>'''))
    init_notebook_mode(connected=False)
```

In [5]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_tra
in_score'], name = 'train')
trace2 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_tes
t_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='Log loss'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



Best Hyperparameters

- Considering the overfitting problem,I thereby chose 64 & 6 for estimators & depth respectively

In [13]:

```
best_parameters = {'n_estimators': [64],'max_depth': [6]}
```

Applying Best Hyperparameters on train & test data & finding the respective log-losses

In [15]:

```
# find train & test log loss for the best hyperparameters [:,1]

xg_best= XGBClassifier(n_estimators= 64 , max_depth= 6)

xg_best.fit(X_tr_tfidf, y_train)

probs_train= xg_best.predict_proba(X_tr_tfidf)
probs_test= xg_best.predict_proba(X_test_tfidf)

y_pred_train= xg_best.predict(X_tr_tfidf)
y_pred_test= xg_best.predict(X_test_tfidf)
```

In [17]:

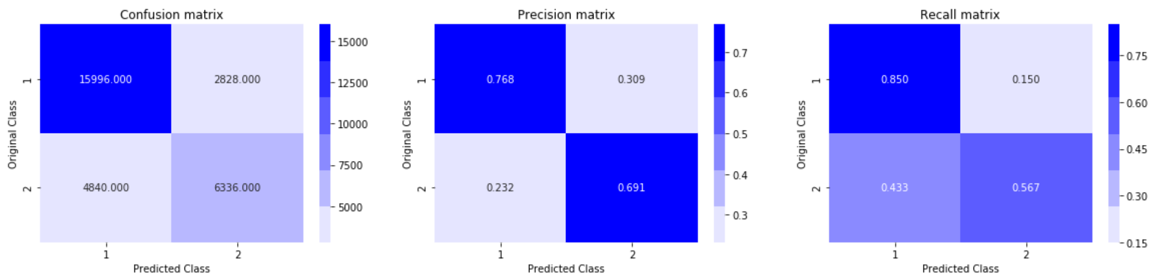
```
print("The train log loss for the best hyperparameters is:",log_loss(y_train, probs_tra
in, eps=1e-15))
print("The test log loss for the best hyperparameters is:",log_loss(y_test, probs_test,
eps=1e-15))
```

```
The train log loss for the best hyperparameters is: 0.49418949719297234
The test log loss for the best hyperparameters is: 0.5037430334584166
```

Plot confusion matrix for test data

In [18]:

```
plot_confusion_matrix(y_test, y_pred_test)
```



6. Featurizing text data using TFIDF W2V & building models

6.1 TFIDF W2V on question columns using a pretrained glove model

In [26]:

```
# pretrained glove model
with open('C:\\Users\\Admin\\Assignments and case studies\\Mandatory\\Assignment 7-SVM
 on donors choose\\glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
print ("Done.",len(model)," words loaded!")
```

```
Done. 51510  words loaded!
```

In [29]:

```
X_train.columns
```

Out[29]:

```
Index(['id', 'question1', 'question2', '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_partial_ratio', 'longest_substr_ratio', 'freq_q
id1',
       'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q
2'],
      dtype='object')
```

Converting the contents to string as there were few float values which were hindering the .split() function

In [38]:

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

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

TFIDF W2V on 'question1' column

In [35]:

```
#fitting the tfidf model only on train data to prevent data leakage

tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['question1'].values.astype('U'))
#we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words_q1 = set(tfidf_model.get_feature_names())
```

In [37]:

```
# For train data

# average Word2Vec using pretrained models
# compute average word2vec for each review.
tfidf_w2v_train_q1 = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['question1']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_q1):
            vec = model[word]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
        tfidf_w2v_train_q1.append(vector)
print(len(tfidf_w2v_train_q1))
print(len(tfidf_w2v_train_q1[0]))
```

```
100%|██████████████████████████████████████████████████
█████████████████████████████████████| 70000/70000 [00:07<0
0:00, 9153.18it/s]

70000
300
```

In [40]:

```
# For test data

tfidf_w2v_test_q1 = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['question1']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_q1):
            vec = model[word]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
        tfidf_w2v_test_q1.append(vector)
print(len(tfidf_w2v_test_q1))
print(len(tfidf_w2v_test_q1[0]))
```

```
100%|██████████████████████████████████████████████████
█████████████████████████████████████| 30000/30000 [00:03<0
0:00, 8870.33it/s]

30000
300
```

TFIDF W2V on 'question2' column

In [42]:

```
#fitting the tfidf model only on train data to prevent data leakage

tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['question2'].values.astype('U'))
#we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words_q2 = set(tfidf_model.get_feature_names())
```

In [49]:

```
# For train data

# average Word2Vec using pretrained models
# compute average word2vec for each review.
tfidf_w2v_train_q2 = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['question2']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_q2):
            vec = model[word]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
        tfidf_w2v_train_q2.append(vector)
print(len(tfidf_w2v_train_q2))
print(len(tfidf_w2v_train_q2[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 70000/70000 [00:07<0
0:00, 8905.04it/s]

70000
300
```

In [47]:

```
# For test data

tfidf_w2v_test_q2 = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['question2']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_q2):
            vec = model[word]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
        tfidf_w2v_test_q2.append(vector)
print(len(tfidf_w2v_test_q2))
print(len(tfidf_w2v_test_q2[0]))
```

```
100%|
| 30000/30000 [00:03<0
0:00, 8949.67it/s]

30000
300
```

6.2 Stacking TFIDF W2V+ Basic + Advanced features

In [64]:

```
#for train data

from scipy.sparse import coo_matrix,hstack

X_tr_tfidf_w2v = hstack((tfidf_w2v_train_q1,tfidf_w2v_train_q2,coo_matrix(X_train_cwc_m
in_norm),coo_matrix(X_train_cwc_max_norm),coo_matrix(X_train_csc_min_norm),coo_matrix(X
_train_csc_max_norm),
                    coo_matrix(X_train_ctc_min_norm),coo_matrix(X_train_ctc_max_norm),c
oo_matrix(X_train_last_word_eq_norm),coo_matrix(X_train_first_word_eq_norm),
                    coo_matrix(X_train_abs_len_diff_norm),coo_matrix(X_train_mean_len_n
orm),coo_matrix(X_train_token_set_ratio_norm),
                    coo_matrix(X_train_token_sort_ratio_norm),coo_matrix(X_train_fuzz_r
atio_norm),coo_matrix(X_train_fuzz_partial_ratio_norm),
                    coo_matrix(X_train_longest_substr_ratio_norm),coo_matrix(X_train_fr
eq_qid1_norm),coo_matrix(X_train_freq_qid2_norm),coo_matrix(X_train_q1len_norm),coo_mat
rix(X_train_q2len_norm),
                    coo_matrix(X_train_q1_n_words_norm),coo_matrix(X_train_q2_n_words_n
orm),coo_matrix(X_train_word_Common_norm),coo_matrix(X_train_word_Total_norm),
                    coo_matrix(X_train_word_share_norm),coo_matrix(X_train_freq_q1addq2
_norm),coo_matrix(X_train_freq_q1subq2_norm))).tocsr()
```

In [74]:

```
#for test data

X_test_tfidf_w2v = hstack((tfidf_w2v_test_q1,tfidf_w2v_test_q2,coo_matrix(X_test_cwc_mi
n_norm),coo_matrix(X_test_cwc_max_norm),coo_matrix(X_test_csc_min_norm),coo_matrix(X_te
st_csc_max_norm),
                    coo_matrix(X_test_ctc_min_norm),coo_matrix(X_test_ctc_max_norm),coo
_matrix(X_test_last_word_eq_norm),coo_matrix(X_test_first_word_eq_norm),
                    coo_matrix(X_test_abs_len_diff_norm),coo_matrix(X_test_mean_len_nor
m),coo_matrix(X_test_token_set_ratio_norm),
                    coo_matrix(X_test_token_sort_ratio_norm),coo_matrix(X_test_fuzz_rat
io_norm),coo_matrix(X_test_fuzz_partial_ratio_norm),
                    coo_matrix(X_test_longest_substr_ratio_norm),coo_matrix(X_test_freq
_qid1_norm),coo_matrix(X_test_freq_qid2_norm),coo_matrix(X_test_q1len_norm),coo_matrix(
X_test_q2len_norm),
                    coo_matrix(X_test_q1_n_words_norm),coo_matrix(X_test_q2_n_words_nor
m),coo_matrix(X_test_word_Common_norm),coo_matrix(X_test_word_Total_norm),
                    coo_matrix(X_test_word_share_norm),coo_matrix(X_test_freq_q1addq2_n
orm),coo_matrix(X_test_freq_q1subq2_norm))).tocsr()
```

In [75]:

```
print("Final Data Matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
Final Data Matrix
(70000, 626) (70000,)
(30000, 626) (30000,)
```

In [76]:

```
# https://stackoverflow.com/questions/8955448/save-load-scipy-sparse-csr-matrix-in-port
able-data-format
from scipy import sparse
sparse.save_npz("X_tr_tfidf_w2v.npz", X_tr_tfidf_w2v)
sparse.save_npz("X_test_tfidf_w2v.npz", X_test_tfidf_w2v)
```

6.3 Confusion matrix, Precision matrix & Recall matrix

In [91]:

```
# 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
ted class j

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

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

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

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

    labels = [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
bels)
    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
bels)
    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
bels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")
```

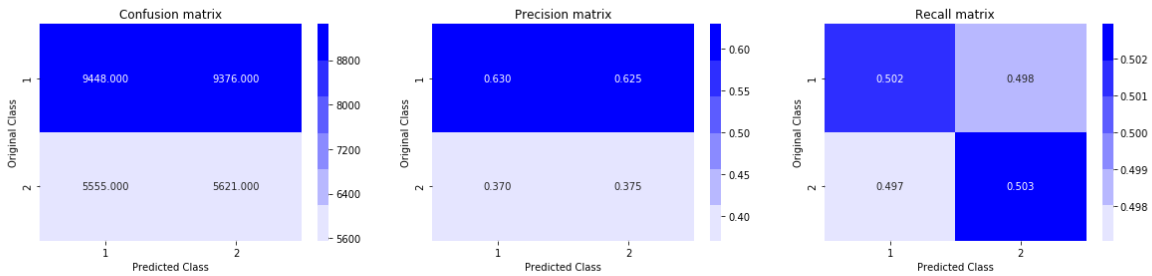
```
    plt.show()
```

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

In [92]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
from sklearn.metrics.classification import accuracy_score, log_loss

test_len = len(y_test)
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    #https://www.geeksforgeeks.org/numpy-random-rand-python/
    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
5))

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

```
Log loss on Test Data using Random Model 0.8803385092512034
```



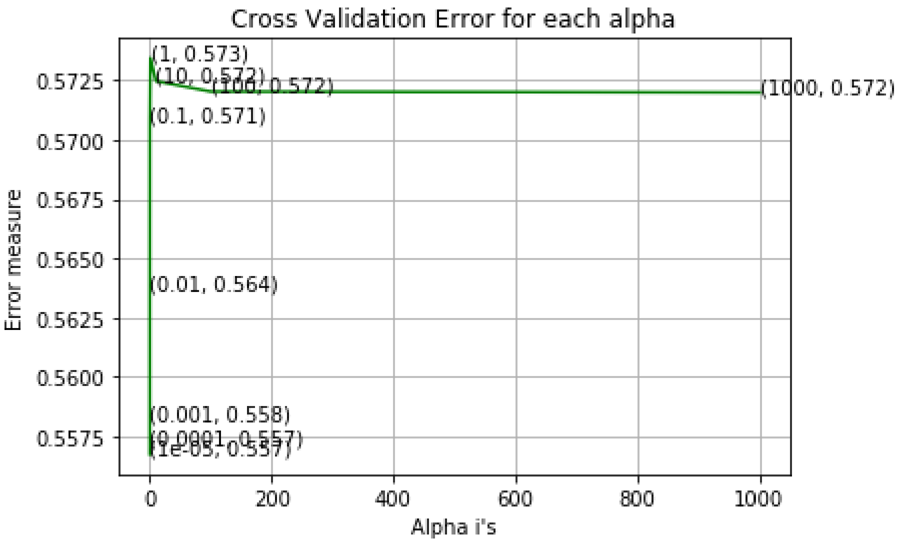
6.5 Logistic Regression with hyperparameter tuning

In [93]:

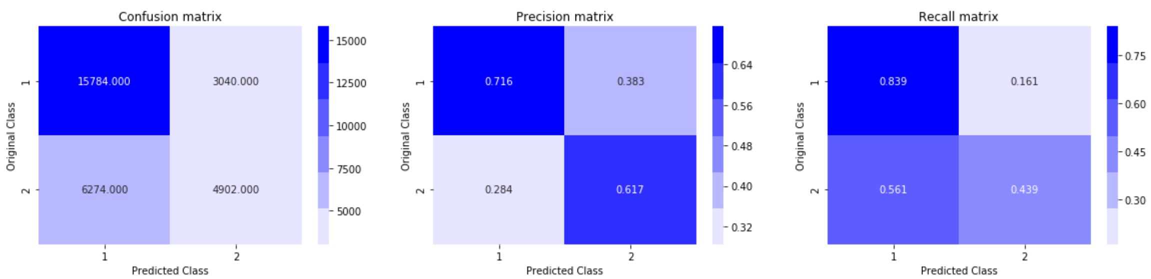
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

alpha = [10 ** x for x in range(-5, 4,)] # hyperparam for SGD classifier.

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

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

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



log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', class_weight='balanced',rand
om_state=42)
    clf.fit(X_tr_tfidf_w2v, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr_tfidf_w2v, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf_w2v)
    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
abels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', class_weight='ba
lanced', random_state=42)
clf.fit(X_tr_tfidf_w2v, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr_tfidf_w2v, y_train)

predict_y = sig_clf.predict_proba(X_tr_tfidf_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_tfidf_w2v)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
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: 0.556740965244953
For values of alpha =  0.0001 The log loss is: 0.5571572922078711
For values of alpha =  0.001 The log loss is: 0.5582202579968012
For values of alpha =  0.01 The log loss is: 0.5636722936390104
For values of alpha =  0.1 The log loss is: 0.5707787427730561
For values of alpha =  1 The log loss is: 0.5734408788711123
For values of alpha =  10 The log loss is: 0.5724647932457096
For values of alpha =  100 The log loss is: 0.5720238972493338
For values of alpha =  1000 The log loss is: 0.5719832437714961
```




```
For values of best alpha =  1e-05 The train log loss is: 0.555158625984446
5
For values of best alpha =  1e-05 The test log loss is: 0.556740965244953
Total number of data points : 30000
```



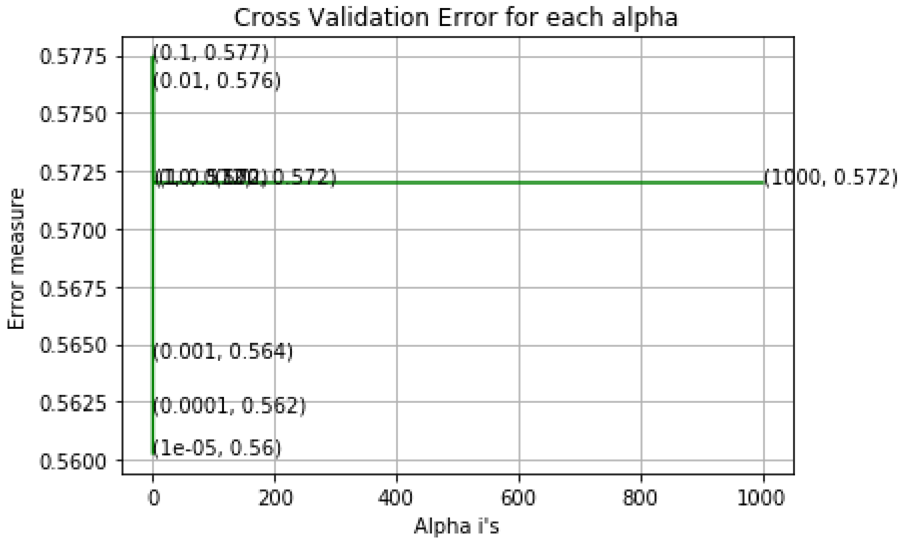
6.6 Linear SVM with hyperparameter tuning

In [94]:

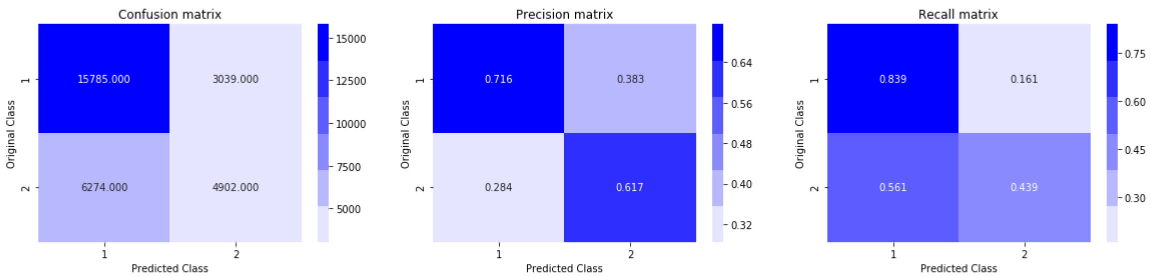
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV

alpha = [10 ** x for x in range(-5, 4,)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', class_weight='balanced',ra
ndom_state=42)
    clf.fit(X_tr_tfidf_w2v, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr_tfidf_w2v, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf_w2v)
    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
abels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', class_weight=
'balanced', random_state=42)
clf.fit(X_tr_tfidf_w2v, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr_tfidf_w2v, y_train)

predict_y = sig_clf.predict_proba(X_tr_tfidf_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_tfidf_w2v)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
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: 0.5602739178758853
For values of alpha =  0.0001 The log loss is: 0.5621231013295959
For values of alpha =  0.001 The log loss is: 0.5644898883770046
For values of alpha =  0.01 The log loss is: 0.5761936770265136
For values of alpha =  0.1 The log loss is: 0.5774482999030066
For values of alpha =  1 The log loss is: 0.572000857193398
For values of alpha =  10 The log loss is: 0.5719951831016798
For values of alpha =  100 The log loss is: 0.571995521788538
For values of alpha =  1000 The log loss is: 0.5719955218377696
```




```
For values of best alpha =  1e-05 The train log loss is: 0.558917466542778
For values of best alpha =  1e-05 The test log loss is: 0.5602739178758853
Total number of data points : 30000
```



6.7 XGBoost with hyperparameter tuning

In [5]:

```
from scipy import sparse
import numpy as np
X_tr_tfidf_w2v= sparse.load_npz("X_tr_tfidf_w2v.npz")
X_test_tfidf_w2v = sparse.load_npz("X_test_tfidf_w2v.npz")
y_train = np.load('y_train.npy')
y_test = np.load('y_test.npy')
```

In [6]:

```
#https://dask-ml.readthedocs.io/en/stable/modules/generated/dask_ml.xgboost.XGBClassifi
er.html
#https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import log_loss
from xgboost import XGBClassifier

xgb = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64],'max_depth': [4, 6, 8, 10, 12]}
clf1 = RandomizedSearchCV(xgb, parameters, cv=5, scoring='neg_log_loss',return_train_sc
ore=True,n_jobs=-1)
rs1 = clf1.fit(X_tr_tfidf_w2v, y_train)
```

In [7]:

```
df=pd.DataFrame(clf1.cv_results_)
df.head(2)
```

Out[7]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max
0	169.665524	6.358930	-0.668843	-0.657575	6
1	169.564930	6.229914	-0.679065	-0.627150	12

2 rows × 22 columns

In [8]:

```
df.to_csv(r'HYP_w2v.csv')
```

In [6]:

```
df = pd.read_csv("HYP_w2v.csv")
```

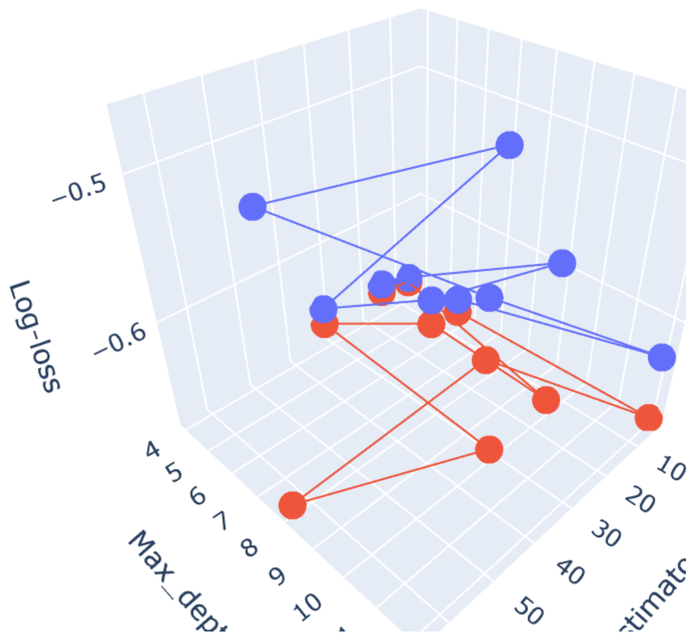
3D-Plot

In [7]:

```
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/requ
ire.js"></script>'''))
    init_notebook_mode(connected=False)
```

In [8]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_tra
in_score'], name = 'train')
trace2 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_tes
t_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='Log-loss'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



Best Hyperparameters

In [14]:

```
print(clf1.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=None, n_estimators=32, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [15]:

```
best_parameters = {'n_estimators': [32],'max_depth': [4]}
```

Applying Best Hyperparameters on train & test data & finding the respective log-losses

In [18]:

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4
params['n_estimators'] = 32

d_train = xgb.DMatrix(X_tr_tfidf_w2v, label=y_train)
d_test = xgb.DMatrix(X_test_tfidf_w2v, label=y_test)

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

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

xgdmat = xgb.DMatrix(X_tr_tfidf_w2v,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y,eps=1e-15))
```

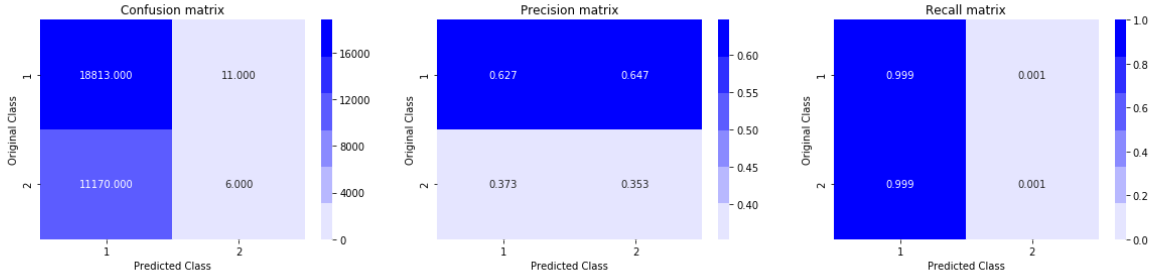
```
[0]     train-logloss:0.691728  valid-logloss:0.6919
Multiple eval metrics have been passed: 'valid-logloss' will be used for e
arly stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.680798  valid-logloss:0.68133
[20]    train-logloss:0.673284  valid-logloss:0.674332
[30]    train-logloss:0.668022  valid-logloss:0.669642
[40]    train-logloss:0.66429   valid-logloss:0.666535
[50]    train-logloss:0.661577  valid-logloss:0.664499
[60]    train-logloss:0.659595  valid-logloss:0.663141
[70]    train-logloss:0.658116  valid-logloss:0.662289
[80]    train-logloss:0.656938  valid-logloss:0.661719
[90]    train-logloss:0.65596   valid-logloss:0.661333
[100]   train-logloss:0.655137  valid-logloss:0.66113
[110]   train-logloss:0.65442   valid-logloss:0.661017
[120]   train-logloss:0.653761  valid-logloss:0.660955
[130]   train-logloss:0.653114  valid-logloss:0.660891
[140]   train-logloss:0.652487  valid-logloss:0.660895
[150]   train-logloss:0.651918  valid-logloss:0.660935
Stopping. Best iteration:
[137]   train-logloss:0.652675  valid-logloss:0.660879

The test log loss is: 0.660925915375352
```

Plot confusion matrix for test data

In [20]:

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

```
Total number of data points : 30000
```



7. Summary of model performances

In [25]:

```
#Ref: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Models","Test Log-loss"]
x.add_row(["TFIDF", "Random model", 0.884])
x.add_row(["TFIDF", "Logistic regression", 0.470])
x.add_row(["TFIDF", "Linear SVM", 0.471])
x.add_row(["TFIDF", "XGBoost", 0.504])

x.add_row(["--------", "------------------", "------------------"])

x.add_row(["TFIDF W2V", "Random model", 0.884])
x.add_row(["TFIDF W2V", "Logistic regression", 0.557])
x.add_row(["TFIDF W2V", "Linear SVM", 0.560])
x.add_row(["TFIDF W2V", "XGBoost", 0.661])
print(x)
```

```
+------------+---------------------+--------------------+
| Vectorizer |        Models       |   Test Log-loss    |
+------------+---------------------+--------------------+
|   TFIDF    |     Random model    |       0.884        |
|   TFIDF    | Logistic regression |        0.47        |
|   TFIDF    |      Linear SVM     |       0.471        |
|   TFIDF    |       XGBoost       |       0.504        |
|  --------  |  ------------------ | ------------------ |
| TFIDF W2V  |     Random model    |       0.884        |
| TFIDF W2V  | Logistic regression |       0.557        |
| TFIDF W2V  |      Linear SVM     |        0.56        |
| TFIDF W2V  |       XGBoost       |       0.661        |
+------------+---------------------+--------------------+
```

8. Conclusion

- From the above table it can be observed that when text data was vectorized using simple TFIDF, linear models (Logistic regression in specific) performed well compared to linear models built on TFIDF W2V. The reason is high dimensionality of the features using TFIDF
- XGBoost could have performed well on TFIDF W2V if more hyperparameters could have been tuned. I just considered max_depth & n_estimators

Procedure followed to solve this case study:

1. The real problem was mapped to a ML problem as a binary classification task where label 1 indicated the question pair was duplicate & 0 indicated thhat the question pair is different from each other.
2. EDA on the data was done where important aspects like output class distribution,total number of Unique Questions, # of unique questions that appear more than once, max number of times a single question was repeated, checking for duplicate question pair,Checking for NULL values in the data & plotting the # of occurances of each question.
3. 11 basic features were constructed using question id's & the question pairs text. word_share & word_common were the features that were analysed in detail using violin plots & pdf in order to determine if they could make an impact in predicting the class labels.
4. 15 advanced nlp features were constructed and then the question pairs were preprocessed to remove html tags, punctuations & stopwords.
5. After preprocessing word clouds for both the class labels were created in order to observe the most frequently occuring words in both the classes
6. token_sort_ratio & fuzz_ratio were analysed using violin plots & pdfs'.
7. All 15 advanced nlp features were visualized using TSNE.
8. Referring to Assignment section-5, the next step was to merge the dataframes of basic & advanced features into a single data frame & then split the data into train & test.
9. Post data split, the text data was vectorized using TFIDF with max features of 3000.
10. Next, all numerical features that included basic & advanced features were scaled using normalization.
11. Once the preprocessing was done, next was to concatenate all features which were in the form of a 2D array using hstack function. In sum, there were 6026 features after concatenation.
12. A random model was built by assigning random probabilities for both the classes using inbuilt random # generator. Log loss was calculated & it was set as a benchmark for other models to perform better. The confusion matrix for test data was plotted in the end.
13. A simple hyperparameter tuned LR model was built and the train,test log loss was observed using the best value of alpha.
14. A linear SVM model was built and hyperparameter tuning was done using simple for loops. Once the best value of alpha was found,the train & test log loss was observed & the confusion matrix was plotted .
15. Finally a hyperparameter tuned XGBoost model was built using n_estimators & max_depth as hyperparameters. Used a 3D plot to determine the best n_estimators & max_depth. The train & test log loss was calculated and the confusion matrix was plotted to observe the FP,TP,FN & TN.
16. Results were tabulated to compare the performance of models.