# 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

____ Useful Links ____ - Discussions : 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 - Blog 1 : 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

### 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 Problem

### 2.1 Data

#### 2.1.1 Data Overview

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

#### 2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in india?","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 happen 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
geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my Youtube
comments?","1"
```

### 2.2 Mapping the real world problem to an ML problem

#### 2.2.1 Type of Machine Leaning Problem

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

### 2.2.2 Performance Metric

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

Metric(s):

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

## 2.3 Train and Test Construction

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

## 3. Exploratory Data Analysis

In [1]:

```
# 1ST ipynb------Quora
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

# 2ND ipynb----------Quora_Preprocessing
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline

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
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

#3rd ipynb--------Q_Mean_W2V
import time
import warnings
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
from tqdm import tqdm

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

#4th ipynb ML_models
import sqlite3
from sqlalchemy import create_engine # database connection
from sqlalchemy import create_engine # database connection
import csv
warnings.filterwarnings("ignore")
import datetime as dt
from sklearn.decomposition import TruncatedSVD
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



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

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

## 3.1 Reading data and basic stats

In [2]:

```
df = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train.csv")

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

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

In [3]:

```
df.head()
```

Out[3]:

| | 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 [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
```

```
 0   id            404290 non-null  int64 
 1   qid1          404290 non-null  int64 
 2   qid2          404290 non-null  int64 
 3   question1     404289 non-null  object
 4   question2     404288 non-null  object
 5   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

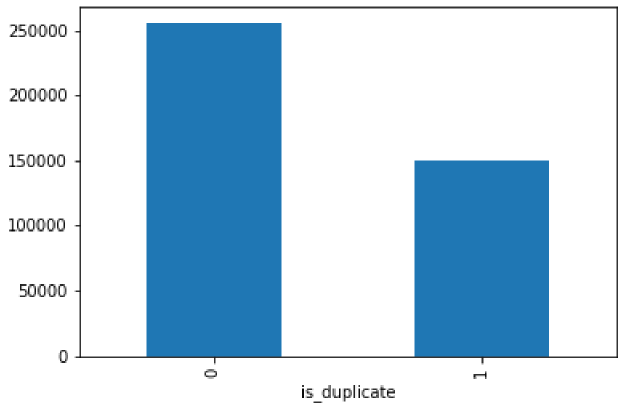
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [5]:

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

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c47d5afc8>
```



In [6]:

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

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

In [7]:

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

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

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

### 3.2.2 Number of unique questions

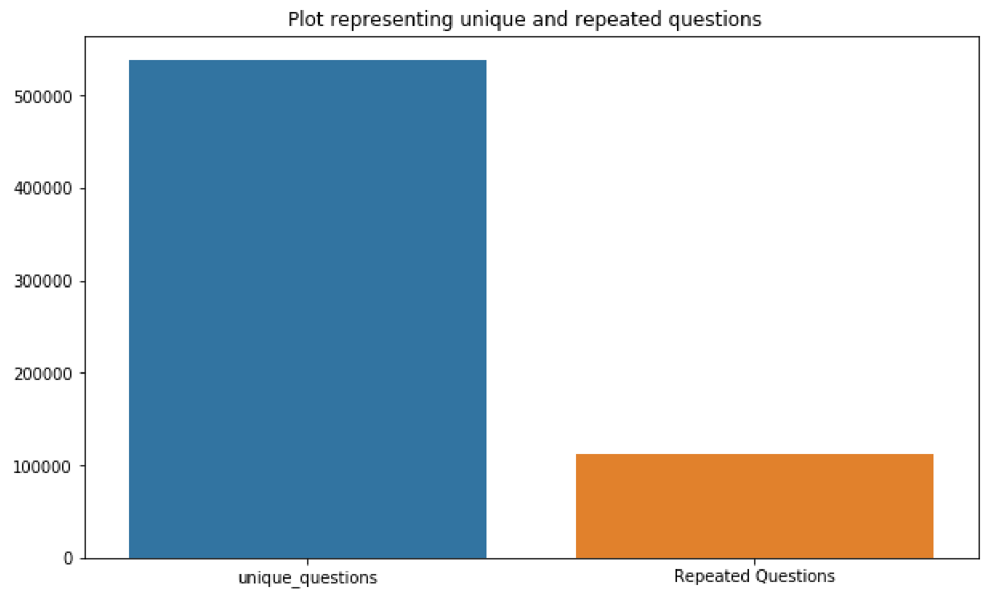
In [8]:

```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {}
({}%)\n'.format(qs_morethan_onetime,qs_morethan_onetime/unique_qs*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 (20.77953945937505%)

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

In [9]:

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

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



### 3.2.3 Checking for Duplicates

In [10]:

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

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

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

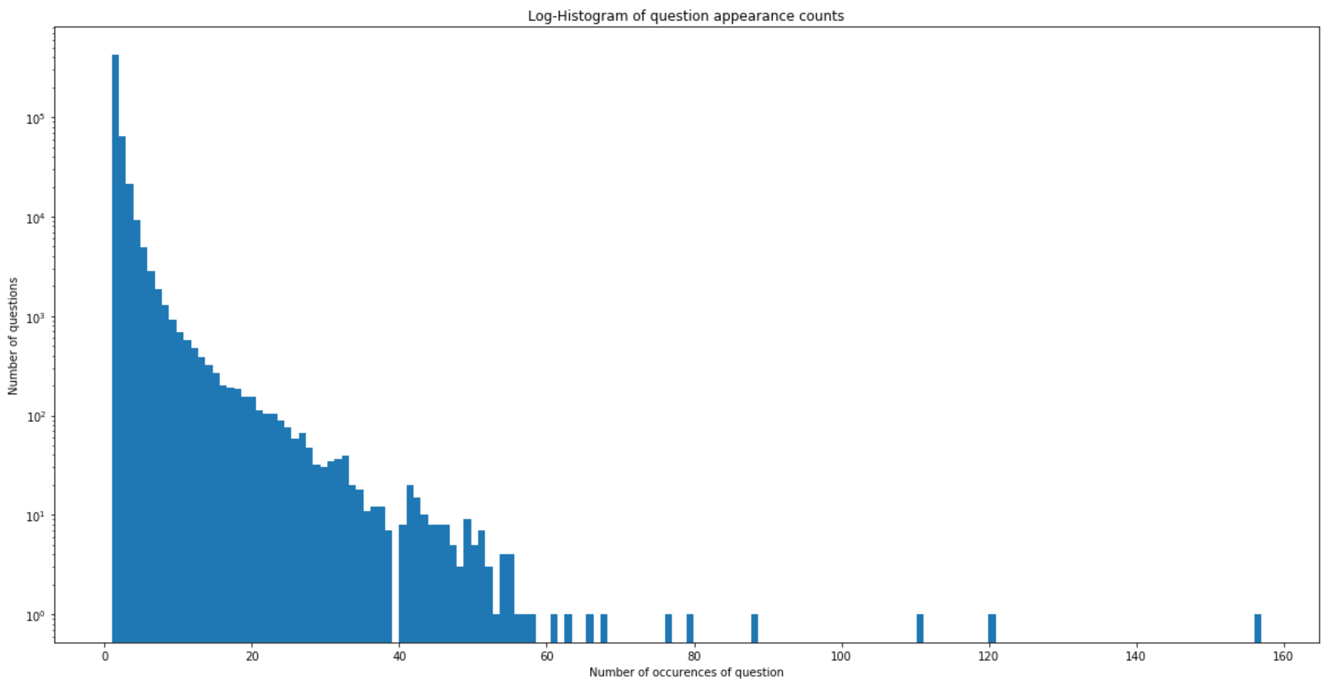
```
Number of duplicate questions 0
```

### 3.2.4 Number of occurrences of each question

In [11]:

```
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.value_counts(
))))
```

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



### 3.2.5 Checking for NULL values

In [12]:

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

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

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

- There are two rows with null values in question2

In [13]:

```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
df=df.sample(n=100000,random_state=1)
df.to_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train.csv")
df.shape
```

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

Out[13]:

```
(100000, 6)
```

## 3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

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

In [14]:

```
if os.path.isfile('E:/BOOKS NEW/Cases datasets/6. Quora Question 
Pairs/df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question 
Pairs/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()
df.head()
```

Out[14]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | What is the step by step guide to invest in sh... | What is the step by step guide to invest in sh... | 0 | 1 | 1 | 66 | 57 | 14 | 12 | 10.0 |
| **1** | 1 | 3 | 4 | What is the story of Kohinoor (Koh-i-Noor) Dia... | What would happen if the Indian government sto... | 0 | 4 | 1 | 51 | 88 | 8 | 13 | 4.0 |
| **2** | 2 | 5 | 6 | How can I increase the speed of my internet co... | How can Internet speed be increased by hacking... | 0 | 1 | 1 | 73 | 59 | 14 | 10 | 4.0 |
| **3** | 3 | 7 | 8 | Why am I mentally very lonely? How can I solve... | Find the remainder when [math]23^{24}[/math] i... | 0 | 1 | 1 | 50 | 65 | 11 | 9 | 0.0 |
| **4** | 4 | 9 | 10 | Which one dissolve in water quikly sugar, salt... | Which fish would survive in salt water? | 0 | 3 | 1 | 76 | 39 | 13 | 7 | 2.0 |

### 3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [15]:

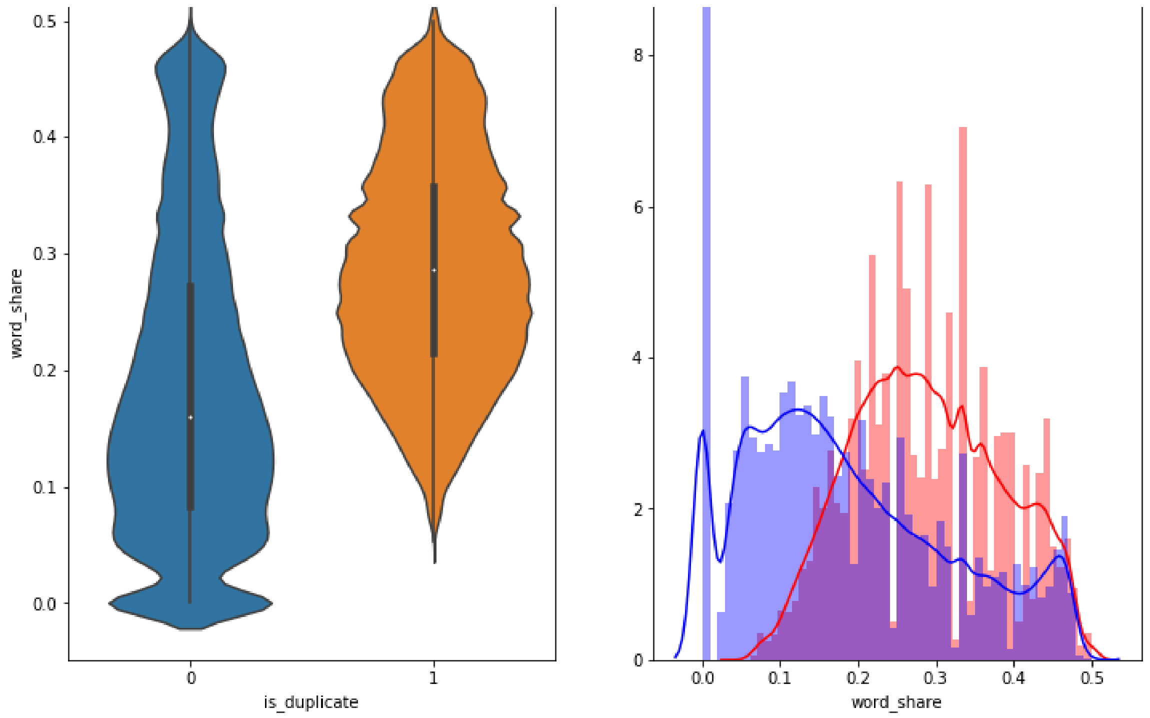
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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 [16]:

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

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

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

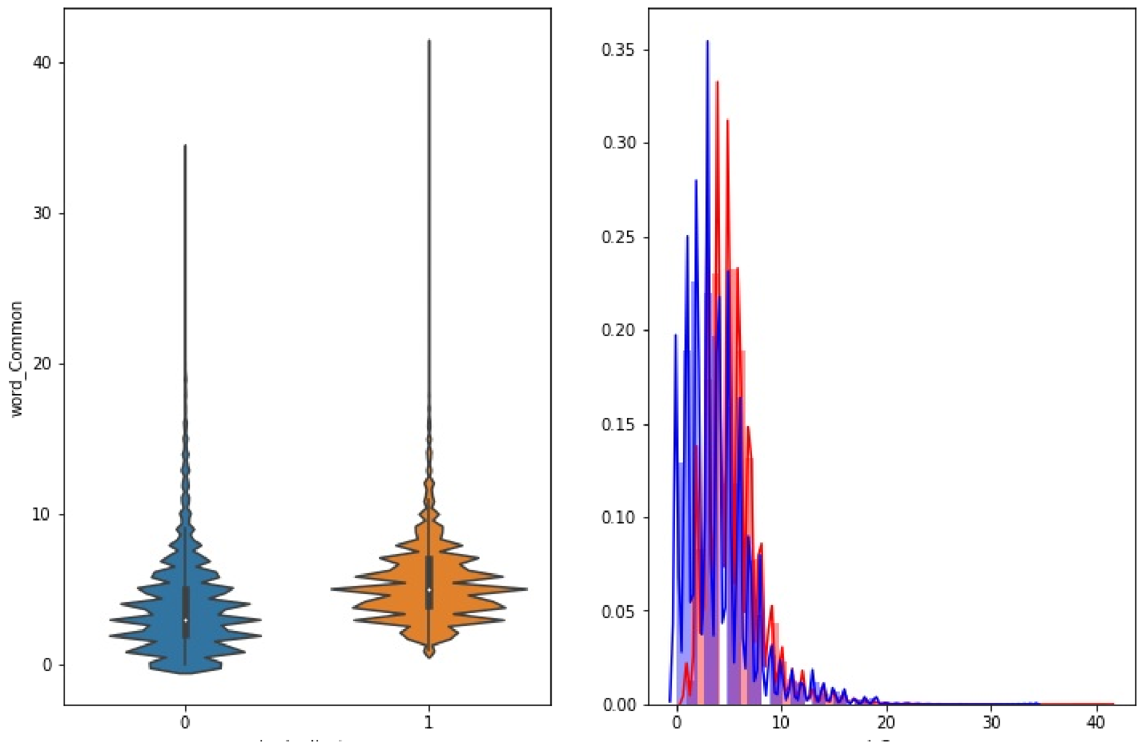


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

#### 3.3.1.2 Feature: word_Common

In [17]:

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

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

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



is_duplicate                                                  word_Common

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

### 1.2.1 : EDA: Advanced Feature Extraction.

In [18]:

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byte-0x9c
if os.path.isfile('E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous notebook")
```

In [19]:

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

Out[19]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common | v |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | What is the step by step guide to invest in sh... | What is the step by step guide to invest in sh... | 0 | 1 | 1 | 66 | 57 | 14 | 12 | 10.0 | |
| **1** | 1 | 3 | 4 | What is the story of Kohinoor (Koh-i-Noor) Dia... | What would happen if the Indian government sto... | 0 | 4 | 1 | 51 | 88 | 8 | 13 | 4.0 | |

## 3.4 Preprocessing of Text

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

In [20]:

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

STOP_WORDS = stopwords.words("english")


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


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

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


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


    return x
```

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

## 3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [21]:

```python
def get_token_features(q1, q2):
    token_features = [0.0]*10
    
    # Converting the Sentence into Tokens: 
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])
    
    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])
    
    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))
    
    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))
    
    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))
    
    
    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    
    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])
    
    # First word of both question is same or not
    token_features[7] = int(q1_tokens[0] == q2_tokens[0])
    
    token_features[8] = abs(len(q1_tokens) - len(q2_tokens))
    
    #Average Token Length of both Questions
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features

# get the Longest Common sub string

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

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

    # Merging Features with dataset

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

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

    #Computing Fuzzy Features and Merging with Dataset

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

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"],
x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the tokens alpha
betically, and
    # then joining them back into a string We then compare the transformed strings with a simple r
atio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"],
x["question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), a
is=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"],
x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["question1"], x["qu
estion2"]), axis=1)
    return df
```

In [22]:

```
if os.path.isfile('E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/nlp_features_train.csv'):
    df = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/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("E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/nlp_features_train.csv", index=
False)
df.head(2)
```

Out[22]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ... | ctc_max | last_word_eq | first_word_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | what is the step by step guide to invest in sh... | what is the step by step guide to invest in sh... | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | ... | 0.785709 | 0.0 | |
| **1** | 1 | 3 | 4 | what is the story of kohinoor koh i noor dia... | what would happen if the indian government sto... | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | ... | 0.466664 | 0.0 | |

2 rows × 21 columns

### 3.5.1 Analysis of extracted features

### 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 [23]:

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

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

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

#Saving the np array into a text file
np.savetxt('E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train_p.txt', p, delimiter=' ', fm
t='%s')
np.savetxt('E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train_n.txt', n, delimiter=' ',  e
ncoding="utf-8", fmt='%s')
```

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

In [24]:

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

textp_w = open(path.join(d, 'E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train_p.txt')).re
ad()
textn_w = open(path.join(d, 'E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train_n.txt')).re
ad()
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 : 33194892
```

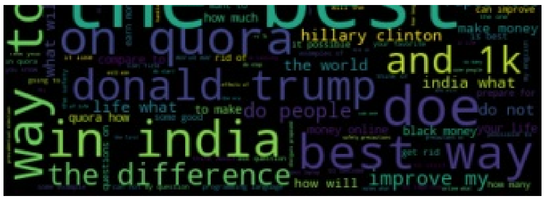
**Word Clouds generated from duplicate pair question's text**

In [25]:

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

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

to learn
difference between
lose weight
ways to
you think
1k rupee
1k note
best book
the best
will be
like to
rupee note



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

In [26]:

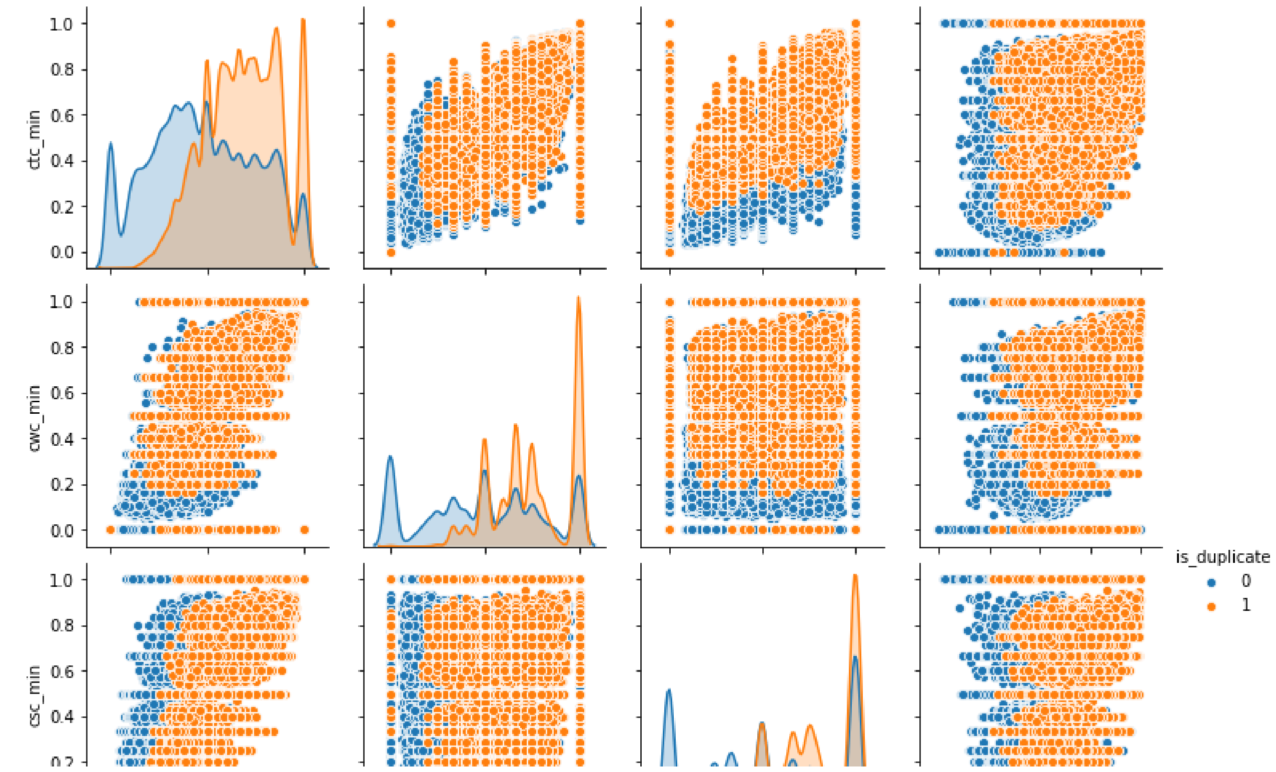
```
wc = WordCloud(background_color="black", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

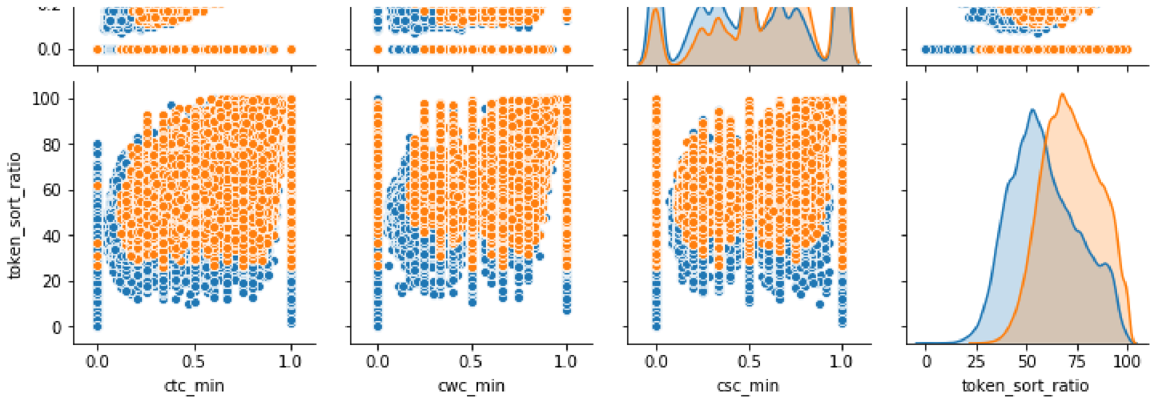
```
Word Cloud for non-Duplicate Question pairs:
```

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

In [27]:

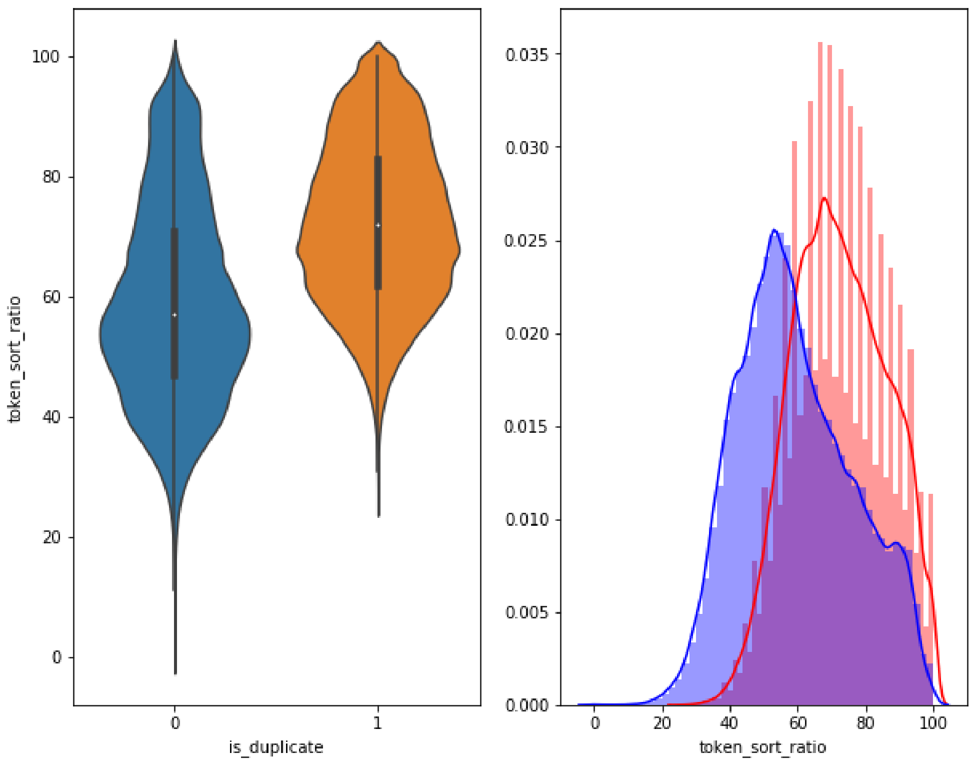
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='i
s_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```





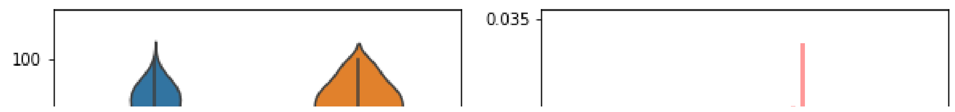
In [28]:

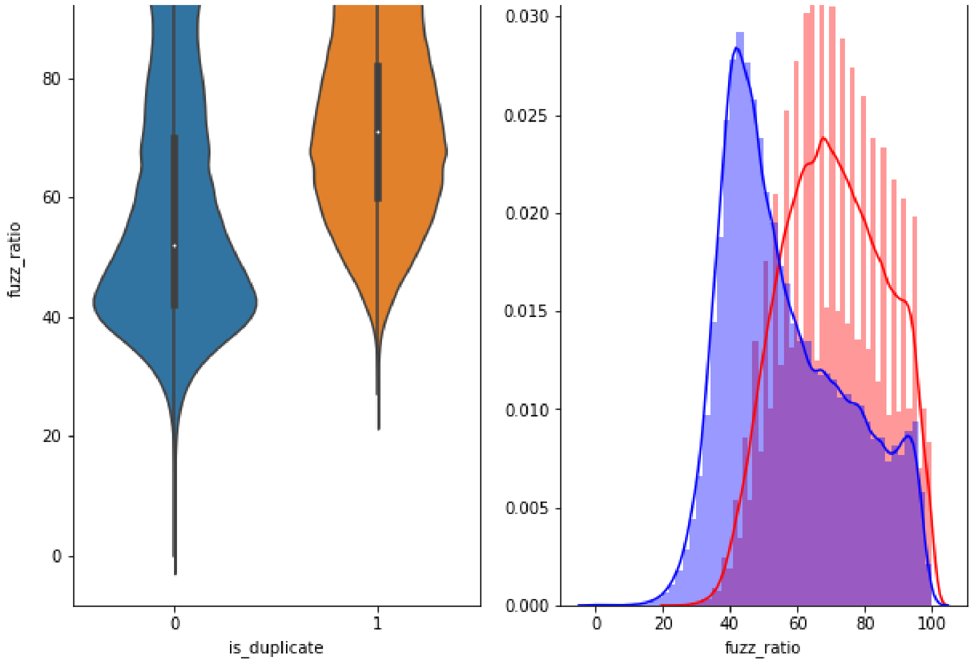
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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" , color = 'blue' )
plt.show()
```



In [29]:

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

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

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





### 3.5.2 Visualization

In [30]:

```
# 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 [31]:

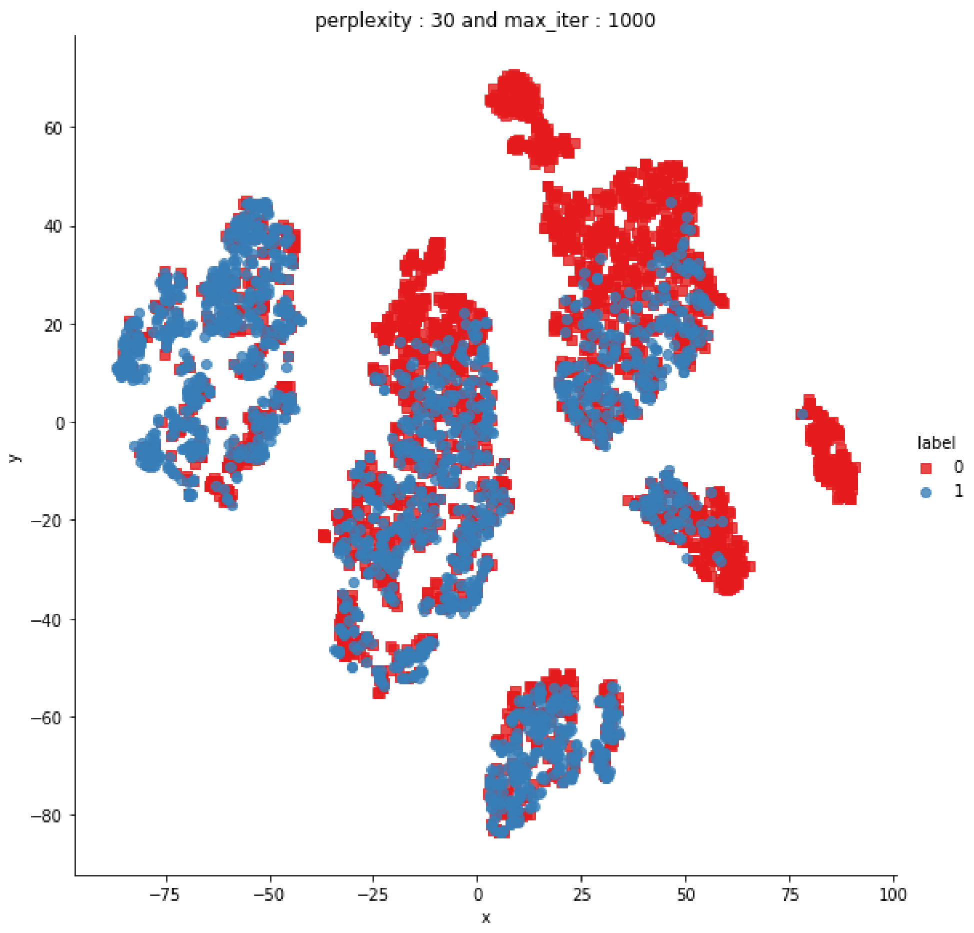
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.023s...
[t-SNE] Computed neighbors for 5000 samples in 0.307s...
[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 0.175s
[t-SNE] Iteration 50: error = 80.9736557, gradient norm = 0.0451379 (50 iterations in 1.254s)
[t-SNE] Iteration 100: error = 70.4410095, gradient norm = 0.0098959 (50 iterations in 1.118s)
[t-SNE] Iteration 150: error = 68.6500015, gradient norm = 0.0059423 (50 iterations in 1.130s)
[t-SNE] Iteration 200: error = 67.8069000, gradient norm = 0.0040715 (50 iterations in 1.169s)
[t-SNE] Iteration 250: error = 67.3088913, gradient norm = 0.0031636 (50 iterations in 1.153s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.308891
[t-SNE] Iteration 300: error = 1.7727009, gradient norm = 0.0011937 (50 iterations in 1.138s)
[t-SNE] Iteration 350: error = 1.3696425, gradient norm = 0.0004815 (50 iterations in 1.106s)
[t-SNE] Iteration 400: error = 1.2022698, gradient norm = 0.0002773 (50 iterations in 1.119s)
[t-SNE] Iteration 450: error = 1.1121849, gradient norm = 0.0001870 (50 iterations in 1.107s)
[t-SNE] Iteration 500: error = 1.0571463, gradient norm = 0.0001402 (50 iterations in 1.120s)
```

```
[t-SNE] Iteration 550: error = 1.0216060, gradient norm = 0.0001162 (50 iterations in 1.229s)
[t-SNE] Iteration 600: error = 0.9982706, gradient norm = 0.0001054 (50 iterations in 1.134s)
[t-SNE] Iteration 650: error = 0.9836186, gradient norm = 0.0000947 (50 iterations in 1.150s)
[t-SNE] Iteration 700: error = 0.9732389, gradient norm = 0.0000854 (50 iterations in 1.133s)
[t-SNE] Iteration 750: error = 0.9652379, gradient norm = 0.0000781 (50 iterations in 1.120s)
[t-SNE] Iteration 800: error = 0.9583344, gradient norm = 0.0000773 (50 iterations in 1.132s)
[t-SNE] Iteration 850: error = 0.9529246, gradient norm = 0.0000718 (50 iterations in 1.124s)
[t-SNE] Iteration 900: error = 0.9486161, gradient norm = 0.0000668 (50 iterations in 1.122s)
[t-SNE] Iteration 950: error = 0.9447117, gradient norm = 0.0000662 (50 iterations in 1.125s)
[t-SNE] Iteration 1000: error = 0.9414362, gradient norm = 0.0000619 (50 iterations in 1.132s)
[t-SNE] KL divergence after 1000 iterations: 0.941436
```

In [32]:

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

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



In [33]:

```python
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 0.018s...
[t-SNE] Computed neighbors for 5000 samples in 0.312s...
```

```
[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 0.175s
[t-SNE] Iteration 50: error = 80.3935394, gradient norm = 0.0316218 (50 iterations in 2.476s)
[t-SNE] Iteration 100: error = 69.1394882, gradient norm = 0.0033516 (50 iterations in 1.806s)
[t-SNE] Iteration 150: error = 67.6482315, gradient norm = 0.0017935 (50 iterations in 1.682s)
[t-SNE] Iteration 200: error = 67.0899506, gradient norm = 0.0012118 (50 iterations in 1.700s)
[t-SNE] Iteration 250: error = 66.7633820, gradient norm = 0.0008854 (50 iterations in 1.684s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.763382
[t-SNE] Iteration 300: error = 1.4975688, gradient norm = 0.0006798 (50 iterations in 1.910s)
[t-SNE] Iteration 350: error = 1.1551027, gradient norm = 0.0001908 (50 iterations in 2.125s)
[t-SNE] Iteration 400: error = 1.0110564, gradient norm = 0.0000958 (50 iterations in 2.093s)
[t-SNE] Iteration 450: error = 0.9386086, gradient norm = 0.0000598 (50 iterations in 2.175s)
[t-SNE] Iteration 500: error = 0.9002966, gradient norm = 0.0000546 (50 iterations in 2.140s)
[t-SNE] Iteration 550: error = 0.8821470, gradient norm = 0.0000461 (50 iterations in 2.130s)
[t-SNE] Iteration 600: error = 0.8714226, gradient norm = 0.0000376 (50 iterations in 2.112s)
[t-SNE] Iteration 650: error = 0.8618767, gradient norm = 0.0000344 (50 iterations in 2.116s)
[t-SNE] Iteration 700: error = 0.8532914, gradient norm = 0.0000330 (50 iterations in 2.124s)
[t-SNE] Iteration 750: error = 0.8468629, gradient norm = 0.0000292 (50 iterations in 2.112s)
[t-SNE] Iteration 800: error = 0.8410668, gradient norm = 0.0000269 (50 iterations in 2.120s)
[t-SNE] Iteration 850: error = 0.8359659, gradient norm = 0.0000300 (50 iterations in 2.103s)
[t-SNE] Iteration 900: error = 0.8317484, gradient norm = 0.0000263 (50 iterations in 2.109s)
[t-SNE] Iteration 950: error = 0.8282196, gradient norm = 0.0000256 (50 iterations in 2.117s)
[t-SNE] Iteration 1000: error = 0.8252402, gradient norm = 0.0000249 (50 iterations in 2.131s)
[t-SNE] KL divergence after 1000 iterations: 0.825240
```

In [34]:

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

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

## 3.6 Featurizing text data with tfidf weighted word-vectors

In [35]:

```
# avoid decoding problems
df = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train.csv")

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

In [36]:

```
df.head()
```

Out[36]:

| | 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 [37]:

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

In [39]:

```
#prepro_features_train.csv (Simple Preprocessing Features)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/nlp_features_train.csv'):
    dfnlp = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/nlp_features_train.csv",encoding='latin-1')
```

```
else:
    print("download nlp_features_train.csv from drive or run previous notebook")

if os.path.isfile('E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [40]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = dfnlp[['id','question1','question2']]
duplicate = dfnlp.is_duplicate
```

In [41]:

```
df3 = df3.fillna(' ')
#assigning new dataframe with columns question(q1+q2) and id same as df3
new_df = pd.DataFrame()
new_df['questions'] = df3.question1 + ' ' + df3.question2
new_df['id'] = df3.id
df2['id']=df1['id']
new_df['id']=df1['id']
final_df = df1.merge(df2, on='id',how='left') #merging df1 and df2
X  = final_df.merge(new_df, on='id',how='left')#merging final_df and new_df
```

In [42]:

```
#removing id from X
X=X.drop('id',axis=1)
X.columns
```

Out[42]:

```
Index(['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_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [43]:

```
y=np.array(duplicate)
```

In [44]:

```
#splitting data into train and test
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=3,test_size=0.3)
```

In [45]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(283003, 27)
(283003,)
(121287, 27)
(121287,)
```

In [46]:

```
#seperating questions for tfidf vectorizer
```

```
X_train_ques=X_train['questions']
X_test_ques=X_test['questions']

X_train=X_train.drop('questions',axis=1)
X_test=X_test.drop('questions',axis=1)
```

In [47]:

```
tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(X_train_ques)

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

## 4.1 Reading data from file and storing into sql table

In [62]:

```
#Creating db file from csv
#if not os.path.isfile('E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train.db'):
#    disk_engine = create_engine('sqlite:///E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/train.db')
#    start = dt.datetime.now()
#    chunksize = 180000
#    j = 0
#    index_start = 1
#    for df in pd.read_csv('E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/final_features.csv', names=['Unnamed:
0','id','is_duplicate','cwc_min','cwc_max','csc_min','csc_max','ctc_min','ctc_max','last_word_eq',
t_word_eq','abs_len_diff','mean_len','token_set_ratio','token_sort_ratio','fuzz_ratio','fuzz_partia
tio','longest_substr_ratio','freq_qid1','freq_qid2','q1len','q2len','q1_n_words','q2_n_words','word
mon','word_Total','word_share','freq_q1+q2','freq_q1-
q2','0_x','1_x','2_x','3_x','4_x','5_x','6_x','7_x','8_x','9_x','10_x','11_x','12_x','13_x','14_x',
x','16_x','17_x','18_x','19_x','20_x','21_x','22_x','23_x','24_x','25_x','26_x','27_x','28_x','29_
0_x','31_x','32_x','33_x','34_x','35_x','36_x','37_x','38_x','39_x','40_x','41_x','42_x','43_x','4
'45_x','46_x','47_x','48_x','49_x','50_x','51_x','52_x','53_x','54_x','55_x','56_x','57_x','58_x',
','60_x','61_x','62_x','63_x','64_x','65_x','66_x','67_x','68_x','69_x','70_x','71_x','72_x','73_x
_x','75_x','76_x','77_x','78_x','79_x','80_x','81_x','82_x','83_x','84_x','85_x','86_x','87_x','88_
89_x','90_x','91_x','92_x','93_x','94_x','95_x','96_x','97_x','98_x','99_x','100_x','101_x','102_x
3_x','104_x','105_x','106_x','107_x','108_x','109_x','110_x','111_x','112_x','113_x','114_x','115_
16_x','117_x','118_x','119_x','120_x','121_x','122_x','123_x','124_x','125_x','126_x','127_x','128_
129_x','130_x','131_x','132_x','133_x','134_x','135_x','136_x','137_x','138_x','139_x','140_x','14
'142_x','143_x','144_x','145_x','146_x','147_x','148_x','149_x','150_x','151_x','152_x','153_x','1
,'155_x','156_x','157_x','158_x','159_x','160_x','161_x','162_x','163_x','164_x','165_x','166_x','
','168_x','169_x','170_x','171_x','172_x','173_x','174_x','175_x','176_x','177_x','178_x','179_x',
x','181_x','182_x','183_x','184_x','185_x','186_x','187_x','188_x','189_x','190_x','191_x','192_x',
_x','194_x','195_x','196_x','197_x','198_x','199_x','200_x','201_x','202_x','203_x','204_x','205_x
6_x','207_x','208_x','209_x','210_x','211_x','212_x','213_x','214_x','215_x','216_x','217_x','218_
19_x','220_x','221_x','222_x','223_x','224_x','225_x','226_x','227_x','228_x','229_x','230_x','231_
232_x','233_x','234_x','235_x','236_x','237_x','238_x','239_x','240_x','241_x','242_x','243_x','24
'245_x','246_x','247_x','248_x','249_x','250_x','251_x','252_x','253_x','254_x','255_x','256_x','2
,'258_x','259_x','260_x','261_x','262_x','263_x','264_x','265_x','266_x','267_x','268_x','269_x','
','271_x','272_x','273_x','274_x','275_x','276_x','277_x','278_x','279_x','280_x','281_x','282_x',
x','284_x','285_x','286_x','287_x','288_x','289_x','290_x','291_x','292_x','293_x','294_x','295_x',
_x','297_x','298_x','299_x','300_x','301_x','302_x','303_x','304_x','305_x','306_x','307_x','308_x
9_x','310_x','311_x','312_x','313_x','314_x','315_x','316_x','317_x','318_x','319_x','320_x','321_
22_x','323_x','324_x','325_x','326_x','327_x','328_x','329_x','330_x','331_x','332_x','333_x','334_
335_x','336_x','337_x','338_x','339_x','340_x','341_x','342_x','343_x','344_x','345_x','346_x','34
'348_x','349_x','350_x','351_x','352_x','353_x','354_x','355_x','356_x','357_x','358_x','359_x','3
,'361_x','362_x','363_x','364_x','365_x','366_x','367_x','368_x','369_x','370_x','371_x','372_x','
','374_x','375_x','376_x','377_x','378_x','379_x','380_x','381_x','382_x','383_x','0_y','1_y','2_y
y','4_y','5_y','6_y','7_y','8_y','9_y','10_y','11_y','12_y','13_y','14_y','15_y','16_y','17_y','18_
19_y','20_y','21_y','22_y','23_y','24_y','25_y','26_y','27_y','28_y','29_y','30_y','31_y','32_y','
,'34_y','35_y','36_y','37_y','38_y','39_y','40_y','41_y','42_y','43_y','44_y','45_y','46_y','47_y',
y','49_y','50_y','51_y','52_y','53_y','54_y','55_y','56_y','57_y','58_y','59_y','60_y','61_y','62_
3_y','64_y','65_y','66_y','67_y','68_y','69_y','70_y','71_y','72_y','73_y','74_y','75_y','76_y','7
'78_y','79_y','80_y','81_y','82_y','83_y','84_y','85_y','86_y','87_y','88_y','89_y','90_y','91_y',
','93_y','94_y','95_y','96_y','97_y','98_y','99_y','100_y','101_y','102_y','103_y','104_y','105_y',
_y','107_y','108_y','109_y','110_y','111_y','112_y','113_y','114_y','115_y','116_y','117_y','118_y
9_y','120_y','121_y','122_y','123_y','124_y','125_y','126_y','127_y','128_y','129_y','130_y','131_
32_y','133_y','134_y','135_y','136_y','137_y','138_y','139_y','140_y','141_y','142_y','143_y','144_
145_y','146_y','147_y','148_y','149_y','150_y','151_y','152_y','153_y','154_y','155_y','156_y','15
'158_y','159_y','160_y','161_y','162_y','163_y','164_y','165_y','166_y','167_y','168_y','169_y','1
'171_y','172_y','173_y','174_y','175_y','176_y','177_y','178_y','179_y','180_y','181_y','182_y','
,'171_y','172_y','173_y','174_y','175_y','176_y','177_y','178_y','179_y','180_y','181_y','182_y','1
','184_y','185_y','186_y','187_y','188_y','189_y','190_y','191_y','192_y','193_y','194_y','195_y',
y','197_y','198_y','199_y','200_y','201_y','202_y','203_y','204_y','205_y','206_y','207_y','208_y',
_y','210_y','211_y','212_y','213_y','214_y','215_y','216_y','217_y','218_y','219_y','220_y','221_y
2_y','223_y','224_y','225_y','226_y','227_y','228_y','229_y','230_y','231_y','232_y','233_y','234_y
35_y','236_y','237_y','238_y','239_y','240_y','241_y','242_y','243_y','244_y','245_y','246_y','247_
248_y','249_y','250_y','251_y','252_y','253_y','254_y','255_y','256_y','257_y','258_y','259_y','26
'261_y','262_y','263_y','264_y','265_y','266_y','267_y','268_y','269_y','270_y','271_y','272_y','2
,'274_y','275_y','276_y','277_y','278_y','279_y','280_y','281_y','282_y','283_y','284_y','285_y','2
','287_y','288_y','289_y','290_y','291_y','292_y','293_y','294_y','295_y','296_y','297_y','298_y',
y','300_y','301_y','302_y','303_y','304_y','305_y','306_y','307_y','308_y','309_y','310_y','311_y',
_y','313_y','314_y','315_y','316_y','317_y','318_y','319_y','320_y','321_y','322_y','323_y','324_y
5_y','326_y','327_y','328_y','329_y','330_y','331_y','332_y','333_y','334_y','335_y','336_y','337_y
38_y','339_y','340_y','341_y','342_y','343_y','344_y','345_y','346_y','347_y','348_y','349_y','350_
351_y','352_y','353_y','354_y','355_y','356_y','357_y','358_y','359_y','360_y','361_y','362_y','36
'364_y','365_y','366_y','367_y','368_y','369_y','370_y','371_y','372_y','373_y','374_y','375_y','3
,'377_y','378_y','379_y','380_y','381_y','382_y','383_y'], chunksize=chunksize, iterator=True, enc
oding='utf-8', ):
#         df.index += index_start
#         j+=1
#         print('{} rows'.format(j*chunksize))
#         df.to_sql('data', disk_engine, if_exists='append')
#         index_start = df.index[-1] + 1
```

In [72]:

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

#    return None


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

In [71]:

```
#read_db = 'E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/train.db'
#conn_r = create_connection(read_db)
#checkTableExists(conn_r)
#conn_r.close()
```

In [70]:

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

#        # for selecting random points
#        data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;", conn_r)
#        #data_tfidf = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 100001;",
conn_r)
#        conn_r.commit()
#        conn_r.close()
```

In [69]:

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

In [68]:

```
#data.head()
```

In [67]:

```
#len(data)
```

## 4.3 Random train test split( 70:30)

In [66]:

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

In [65]:

```
#x_train,x_test, Y_train, Y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

In [64]:

```
#print("Number of data points in train data :",x_train.shape)
#print("Number of data points in test data :",x_test.shape)
```

In [63]:

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

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

In [48]:

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
#for train dataset

nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_train_ques)):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
#df['q1_feats_m'] = list(vecs1)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 283003/283003
[33:22<00:00, 141.36it/s]
```

In [49]:

```
vecs2 = []
for qu2 in tqdm(list(X_test_ques)):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 96])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
#df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 121287/121287
[14:33<00:00, 138.84it/s]
```

In [50]:

```
first_df=pd.DataFrame(vecs1)
sec_df=pd.DataFrame(vecs2)
```

In [51]:

```
X_train = hstack((X_train.values,first_df))
X_test= hstack((X_test.values,sec_df))
print(X_train.shape)
print(X_test.shape)
```

```
(283003, 122)
(121287, 122)
```

## 4. Machine Learning Models

In [52]:

```
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 : (283003, 122)
Number of data points in test data : (121287, 122)
```

In [53]:

```
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:  0.6296541026066862 Class 1:  0.37034589739331386
---------- Distribution of output variable in train data ----------
Class 0:  0.3665190828365777 Class 1:  0.3665190828365777
```

## 4.2 Converting strings to numerics

In [60]:

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

In [61]:

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

In [54]:

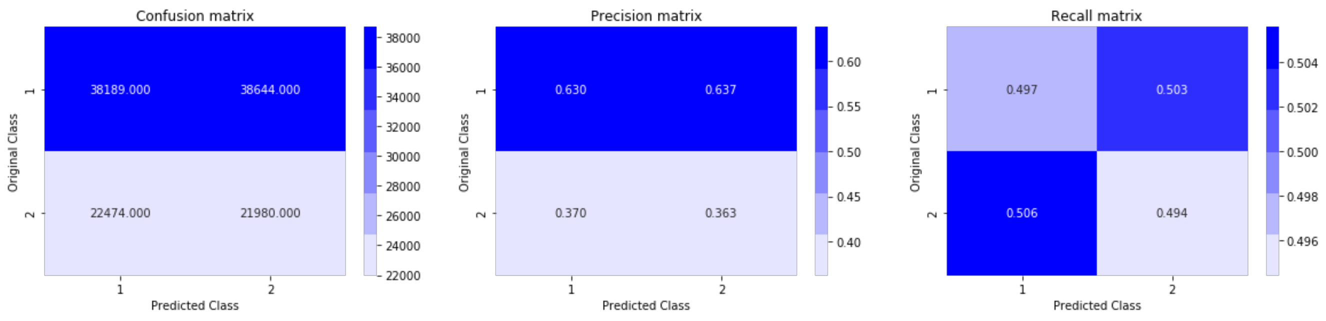
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column
    
    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two 
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1
    
    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two 
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                       [3/4, 4/6]]
    plt.figure(figsize=(20,4))
    
    labels = [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=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")
    
    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

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

    plt.show()
```

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

In [55]:

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

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

```
Log loss on Test Data using Random Model 0.8930710772308218
```



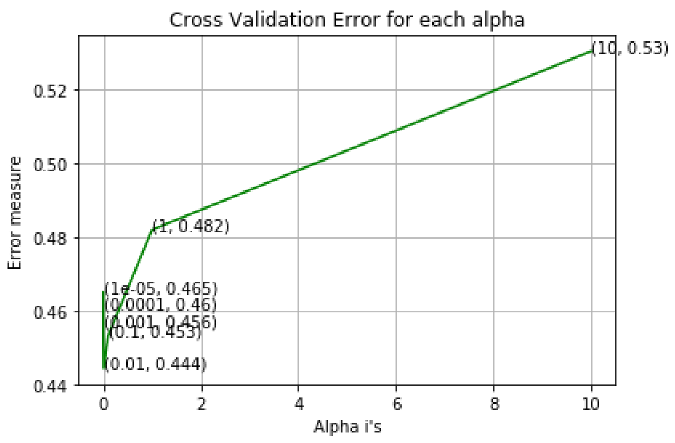
## 4.4 Logistic Regression with hyperparameter tuning

In [56]:

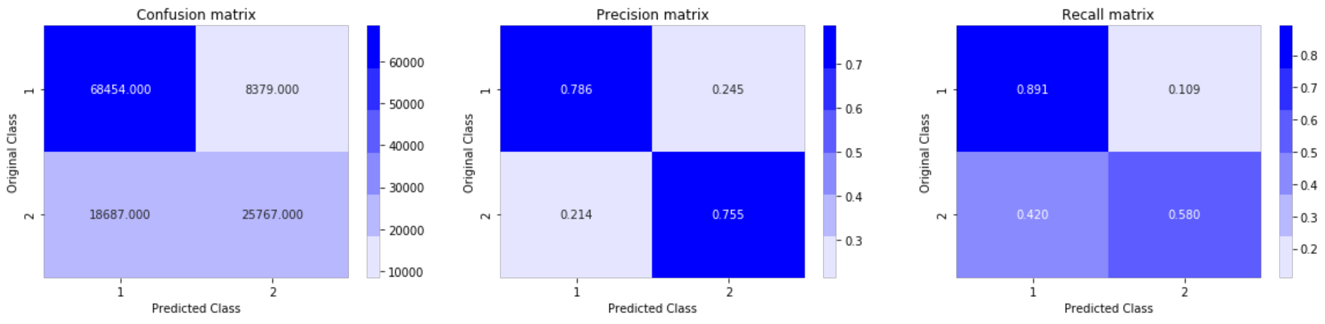
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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


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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_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)
```

```
 14%|███████                                               | 1/7
[15:19<1:31:55, 919.24s/it]

For values of alpha =  1e-05 The log loss is: 0.4648312463166308

 29%|██████████████                                        | 2/7 [31:00<
17:09, 925.92s/it]

For values of alpha =  0.0001 The log loss is: 0.46031375594672336

 43%|██████████████████████                                 | 3/7
[43:27<58:08, 872.23s/it]

For values of alpha =  0.001 The log loss is: 0.4555067003432837

 57%|█████████████████████████████                          | 4/7 [48:2
<35:01, 700.55s/it]

For values of alpha =  0.01 The log loss is: 0.4442784682687898

 71%|████████████████████████████████████                   | 5/7
[50:02<17:17, 518.95s/it]

For values of alpha =  0.1 The log loss is: 0.45309793205401766

 86%|███████████████████████████████████████████            | 6/7 [50:3
8<06:14, 374.06s/it]

For values of alpha =  1 The log loss is: 0.48179960593230503

100%|██████████████████████████████████████████████████████| 7/7 [51:
00<00:00, 437.26s/it]
```

```
For values of alpha =  10 The log loss is: 0.530192539881724
```



```
For values of best alpha =  0.01 The train log loss is: 0.44513853307858486
For values of best alpha =  0.01 The test log loss is: 0.4442784682687898
Total number of data points : 121287
```



## 4.5 Linear SVM with hyperparameter tuning

In [57]:

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

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

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

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


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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_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)
```

```
 14%|███████████                                                                      | 1/7
[18:36<1:51:36, 1116.15s/it]
```

```
For values of alpha =  1e-05 The log loss is: 0.6571085741821002
```

```
 29%|███████████████████████                                                          | 2/7 [36:36<1
2:06, 1105.30s/it]
```

```
For values of alpha =  0.0001 The log loss is: 0.6571085741821002
```

```
 43%|███████████████████████████████████                                              | 3/7
[54:16<1:12:46, 1091.72s/it]
```

```
For values of alpha =  0.001 The log loss is: 0.5100275749716494
```

```
 57%|██████████████████████████████████████████████                                   | 4/7 [1:12:24
54:31, 1090.56s/it]
```

```
For values of alpha =  0.01 The log loss is: 0.4902036861337324
```

```
 71%|██████████████████████████████████████████████████████████                       | 5/7
[1:32:45<37:39, 1129.96s/it]
```

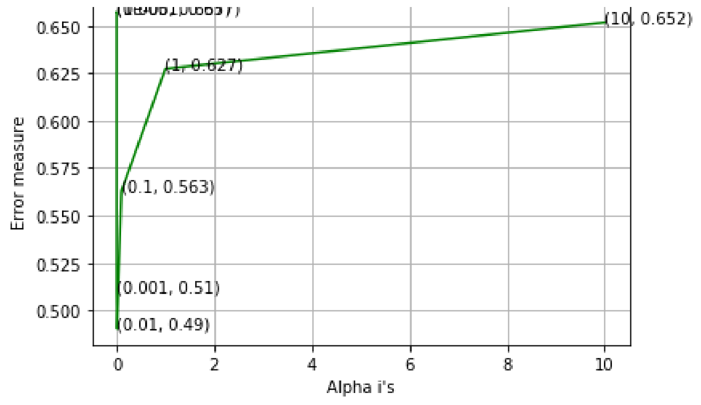
```
For values of alpha =  0.1 The log loss is: 0.5627519283277457
```

```
 86%|██████████████████████████████████████████████████████████████████████           | 6/7 [1:53:17
<19:20, 1160.33s/it]
```

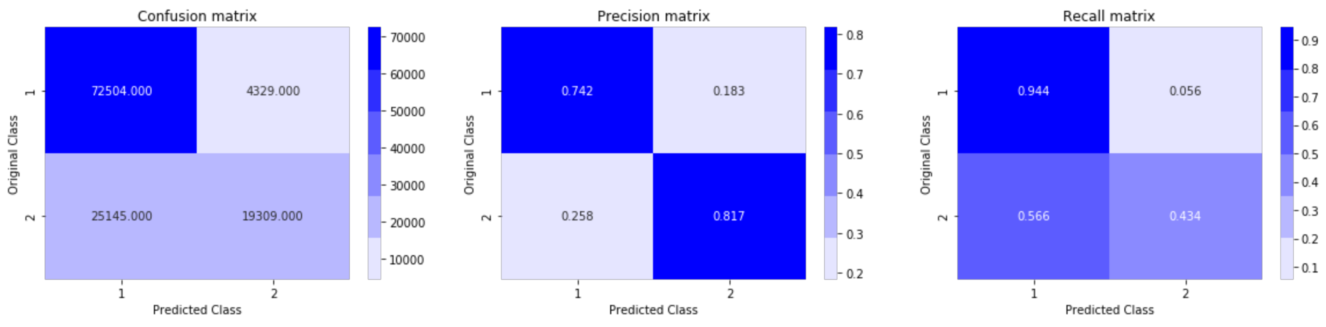
```
For values of alpha =  1 The log loss is: 0.6273330604180539
```

```
100%|█████████████████████████████████████████████████████████████████████████████████| 7/7
[2:15:04<00:00, 1157.80s/it]
```

```
For values of alpha =  10 The log loss is: 0.6516720262639343
```



```
For values of best alpha =  0.01 The train log loss is: 0.4932394282030793
For values of best alpha =  0.01 The test log loss is: 0.4902036861337324
Total number of data points : 121287
```



## 4.2 Converting strings to numerics

## 4.6 XGBoost

In [58]:

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

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

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

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

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

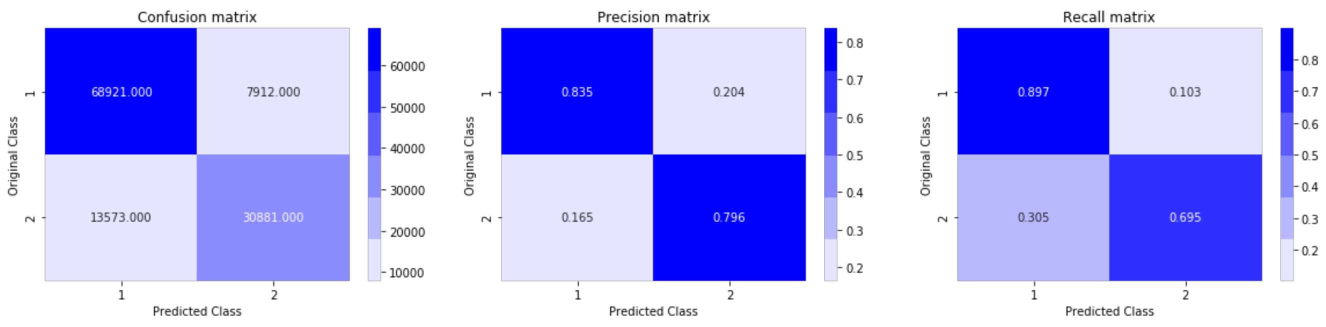
```
[0] train-logloss:0.684841 valid-logloss:0.684894
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10] train-logloss:0.615023 valid-logloss:0.615258
[20] train-logloss:0.564362 valid-logloss:0.564692
[30] train-logloss:0.526544 valid-logloss:0.526955
[40] train-logloss:0.497066 valid-logloss:0.497491
[50] train-logloss:0.474036 valid-logloss:0.47455
[60] train-logloss:0.45565 valid-logloss:0.456199
[70] train-logloss:0.440968 valid-logloss:0.441597
[80] train-logloss:0.429059 valid-logloss:0.429749
[90] train-logloss:0.419548 valid-logloss:0.420221
[100] train-logloss:0.411418 valid-logloss:0.412163
[110] train-logloss:0.404748 valid-logloss:0.405541
```

```
[120] train-logloss:0.399063 valid-logloss:0.399911
[130] train-logloss:0.394362 valid-logloss:0.395266
[140] train-logloss:0.390414 valid-logloss:0.39141
[150] train-logloss:0.387193 valid-logloss:0.388289
[160] train-logloss:0.384286 valid-logloss:0.385417
[170] train-logloss:0.381558 valid-logloss:0.38279
[180] train-logloss:0.379073 valid-logloss:0.380375
[190] train-logloss:0.377047 valid-logloss:0.378434
[200] train-logloss:0.375158 valid-logloss:0.37662
[210] train-logloss:0.373441 valid-logloss:0.374955
[220] train-logloss:0.371786 valid-logloss:0.373406
[230] train-logloss:0.370238 valid-logloss:0.371953
[240] train-logloss:0.368674 valid-logloss:0.370483
[250] train-logloss:0.367354 valid-logloss:0.369234
[260] train-logloss:0.36596 valid-logloss:0.367944
[270] train-logloss:0.364684 valid-logloss:0.36677
[280] train-logloss:0.363422 valid-logloss:0.365605
[290] train-logloss:0.362175 valid-logloss:0.364444
[300] train-logloss:0.361136 valid-logloss:0.363491
[310] train-logloss:0.360112 valid-logloss:0.362578
[320] train-logloss:0.35918 valid-logloss:0.361729
[330] train-logloss:0.358184 valid-logloss:0.360812
[340] train-logloss:0.357075 valid-logloss:0.359765
[350] train-logloss:0.35622 valid-logloss:0.358993
[360] train-logloss:0.355306 valid-logloss:0.358153
[370] train-logloss:0.354462 valid-logloss:0.357392
[380] train-logloss:0.353726 valid-logloss:0.356746
[390] train-logloss:0.352998 valid-logloss:0.356089
[399] train-logloss:0.352263 valid-logloss:0.355446
The test log loss is: 0.3554456865287508
```

In [59]:

```
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 : 121287
```



## 5. Assignments

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Perform hyperparameter tuning of XgBoost models using RandomsearchCV with vectorizer as TF-IDF W2V to reduce the log-loss.

https://github.com/krpiyush5/Quora-Question-Pair-Similarity-Problem-/blob/master/final_update_QuoraQuestionPair.ipynb

## TFIDF instead of TFIDFW2V

In [5]:

```
dfnlp_tfidf = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question Pairs/nlp_features_train
- Copy.csv",encoding='latin-1')
dfppro_tfidf = pd.read_csv("E:/BOOKS NEW/Cases datasets/6. Quora Question
Pairs/df_fe_without_preprocessing_train - Copy.csv",encoding='latin-1')
df1_tfidf = dfnlp_tfidf.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df2_tfidf = dfppro_tfidf.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

```
df3_tfidf = dfnlp_tfidf[['id','question1','question2']]
duplicate_tfidf = dfnlp_tfidf.is_duplicate
```

In [6]:

```
#df1_tfidf=df1_tfidf.drop('Unnamed: 0',axis=1)
df3_tfidf = df3_tfidf.fillna(' ')
#assigning new dataframe with columns question(q1+q2) and id same as df3
new_df_tfidf = pd.DataFrame()
new_df_tfidf['questions'] = df3_tfidf.question1 + ' ' + df3_tfidf.question2
new_df_tfidf['id'] = df3_tfidf.id
df2_tfidf['id']=df1_tfidf['id']
new_df_tfidf['id']=df1_tfidf['id']
final_df_tfidf = df1_tfidf.merge(df2_tfidf, on='id',how='left') #merging df1 and df2
X_tfidf  = final_df_tfidf.merge(new_df_tfidf, on='id',how='left')#merging final_df and new_df
```

In [7]:

```
#removing id from X
X_tfidf = X_tfidf.drop('id',axis=1)
X_tfidf.columns
```

Out[7]:

```
Index(['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_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [8]:

```
y_tfidf=np.array(duplicate_tfidf)
```

In [9]:

```
X_train,X_test, y_train, y_test = train_test_split(X_tfidf, y_tfidf, stratify=y_tfidf, test_size=0.
3)
```

In [10]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(283003, 27)
(283003,)
(121287, 27)
(121287,)
```

In [11]:

```
#seperating questions for tfidf vectorizer
X_train_ques=X_train['questions']
X_test_ques=X_test['questions']

X_train=X_train.drop('questions',axis=1)
X_test=X_test.drop('questions',axis=1)
```

In [12]:

```
len(X_train_ques)
```

Out[12]:

```
283003
```

In [13]:

```
len(X_test_ques)
```

Out[13]:

```
121287
```

In [14]:

```
#tfidf vectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,3),min_df=10)
X_train_tfidf=tf_idf_vect.fit_transform(X_train_ques)
X_test_tfidf=tf_idf_vect.transform(X_test_ques)
```

In [15]:

```
#adding tfidf features to our train and test data using hstack
X_train = hstack((X_train.values,X_train_tfidf))
X_test= hstack((X_test.values,X_test_tfidf))
print(X_train.shape)
print(X_test.shape)
```

```
(283003, 122882)
(121287, 122882)
```

## Applying Logistic Regression

In [85]:

```
alpha = [10 ** x for x in range(-5, 3)] # hyperparam for SGD classifier.

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

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

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


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

```python
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-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)
```

```
  0%|                                                                     |
[00:00<?, ?it/s]

 12%|██████████                                                    | 1/8
[06:27<45:15, 387.98s/it]
```

```
For values of alpha =  1e-05 The log loss is: 0.37972555220565674
```

```
 25%|█████████████████████                                         | 2/8 [09:3
32:47, 327.91s/it]
```

```
For values of alpha =  0.0001 The log loss is: 0.39065733329766117
```

```
 38%|███████████████████████████████                               | 3/8
[10:32<20:33, 246.63s/it]
```

```
For values of alpha =  0.001 The log loss is: 0.42456298372546564
```

```
 50%|██████████████████████████████████████████                    | 4/8 [10:5
<11:55, 178.83s/it]
```

```
For values of alpha =  0.01 The log loss is: 0.4441093313954125
```

```
 62%|████████████████████████████████████████████████████          | 5/8 [11:0
<06:25, 128.51s/it]
```

```
For values of alpha =  0.1 The log loss is: 0.4612945842319354
```

```
 75%|███████████████████████████████████████████████████████████████ | 6/8 [11:
11<03:04, 92.20s/it]
```

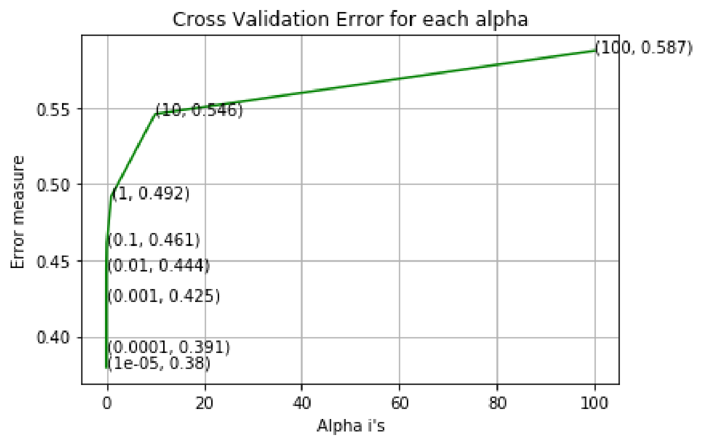
```
For values of alpha =  1 The log loss is: 0.4917544240478566
```

```
 88%|████████████████████████████████████████████                 | 7/8 [11:
18<01:06, 66.56s/it]
```

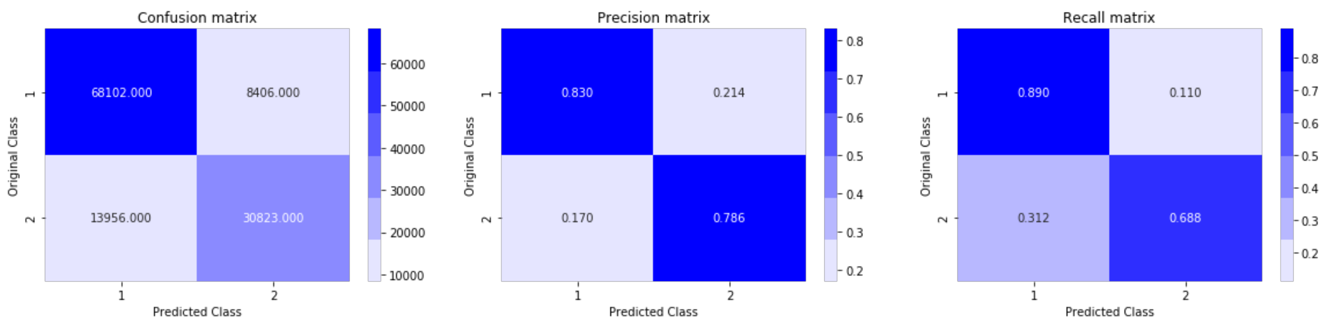
```
For values of alpha =  10 The log loss is: 0.5457386280749916
```

```
100%|████████████████████████████████████████████████████████████| 8/8 [11
:25<00:00, 85.64s/it]
```

```
For values of alpha =  100 The log loss is: 0.5871851308456303
```



```
For values of best alpha =  1e-05 The train log loss is: 0.37721681373075944
For values of best alpha =  1e-05 The test log loss is: 0.37972555220565674
Total number of data points : 121287
```



## Applying Linear SVM

In [86]:

```
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/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_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)
```

```
  0%|                                                                                     |
[00:00<?, ?it/s]


 11%|█████████                                                                  | 1/9
[04:43<37:47, 283.38s/it]
```

```
For values of alpha =  1e-05 The log loss is: 0.414170284719538
```

```
 22%|██████████████████                                                         | 2/9
[08:45<31:37, 271.01s/it]
```

```
For values of alpha =  0.0001 The log loss is: 0.4352724795930874
```

```
 33%|███████████████████████████                                                | 3/9 [14:3
```

```
29:33, 295.65s/it]
```

```
For values of alpha =  0.001 The log loss is: 0.466217775238998
```

```
 44%|██████████████████████████████████████▌                                                 | 4/9
[19:55<25:09, 301.93s/it]
```

```
For values of alpha =  0.01 The log loss is: 0.4771702137147845
```

```
 56%|████████████████████████████████████████████████                                        | 5/9 [23:2
<18:21, 275.29s/it]
```

```
For values of alpha =  0.1 The log loss is: 0.5052846262413879
```

```
 67%|█████████████████████████████████████████████████████████▉                              | 6/9
[26:48<12:38, 252.85s/it]
```

```
For values of alpha =  1 The log loss is: 0.5717937361519422
```

```
 78%|███████████████████████████████████████████████████████████████████▌                    | 7/9 [26:5
6<05:58, 179.32s/it]
```

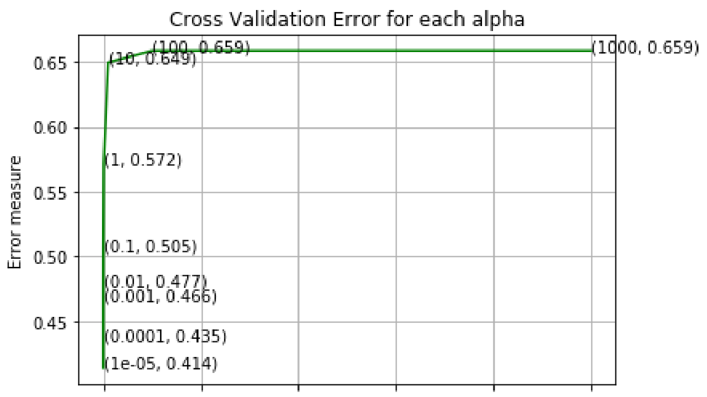
```
For values of alpha =  10 The log loss is: 0.6494222101132184
```

```
 89%|█████████████████████████████████████████████████████████████████████████████▏          | 8/9 [27:0
3<02:07, 127.65s/it]
```

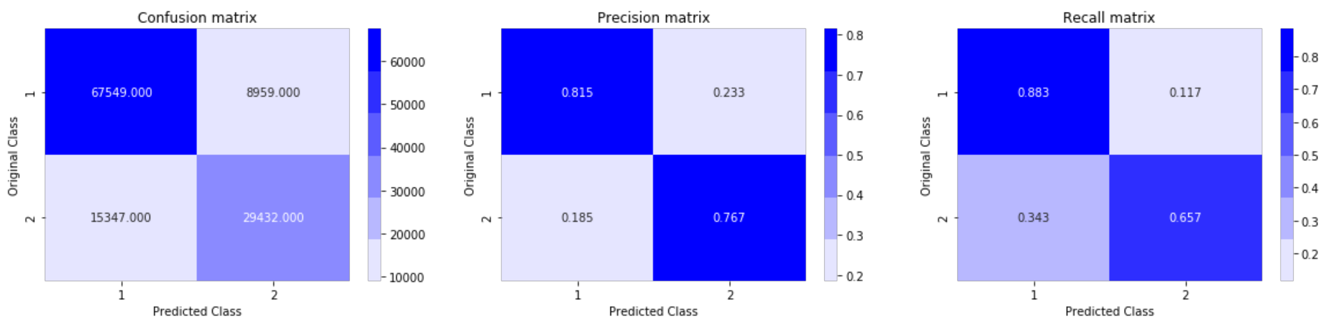
```
For values of alpha =  100 The log loss is: 0.6585278256322703
```

```
100%|████████████████████████████████████████████████████████████████████████████████████████| 9/9 [27:
10<00:00, 181.19s/it]
```

```
For values of alpha =  1000 The log loss is: 0.6585278256322703
```



```
For values of best alpha =  1e-05 The train log loss is: 0.4137798963588442
For values of best alpha =  1e-05 The test log loss is: 0.414170284719538
Total number of data points : 121287
```



## XGBOOST

In [16]:

```
import xgboost as xgb
n_estimators = [50,100,150,200,300,400,500]
test_scores = []
train_scores = []
for i in tqdm(n_estimators):
    clf = xgb.XGBClassifier(learning_rate=0.1,n_estimators=i,n_jobs=-1)
    clf.fit(X_train,y_train)
    y_pred = clf.predict_proba(X_train)
    log_loss_train = log_loss(y_train, y_pred, eps=1e-15)
    train_scores.append(log_loss_train)
    y_pred = clf.predict_proba(X_test)
    log_loss_test = log_loss(y_test, y_pred, eps=1e-15)
    test_scores.append(log_loss_test)
    print('For n_estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
```

```
 14%|██████                                                  | 1/7
[00:26<02:36, 26.10s/it]

For n_estimators =  50 Train Log Loss  0.38002831026556777 Test Log Loss  0.38165567051146976

 29%|████████████                                            | 2/7 [01:
<02:33, 30.75s/it]

For n_estimators =  100 Train Log Loss  0.3596764938859856 Test Log Loss  0.36230325315057776

 43%|██████████████████                                      | 3/7
[02:03<02:33, 38.32s/it]

For n_estimators =  150 Train Log Loss  0.35043622821075626 Test Log Loss  0.3539232670399379

 57%|████████████████████████                                | 4/7 [03:
4<02:23, 47.96s/it]

For n_estimators =  200 Train Log Loss  0.34334442538449134 Test Log Loss  0.34762425259582574

 71%|██████████████████████████████                          | 5/7
[04:49<02:04, 62.15s/it]

For n_estimators =  300 Train Log Loss  0.3334650713013348 Test Log Loss  0.33924066895086424

 86%|████████████████████████████████████                    | 6/7 [06:
52<01:20, 80.38s/it]
```

```
For n_estimators =  400 Train Log Loss  0.32638037580209805 Test Log Loss  0.33352555629587766
```

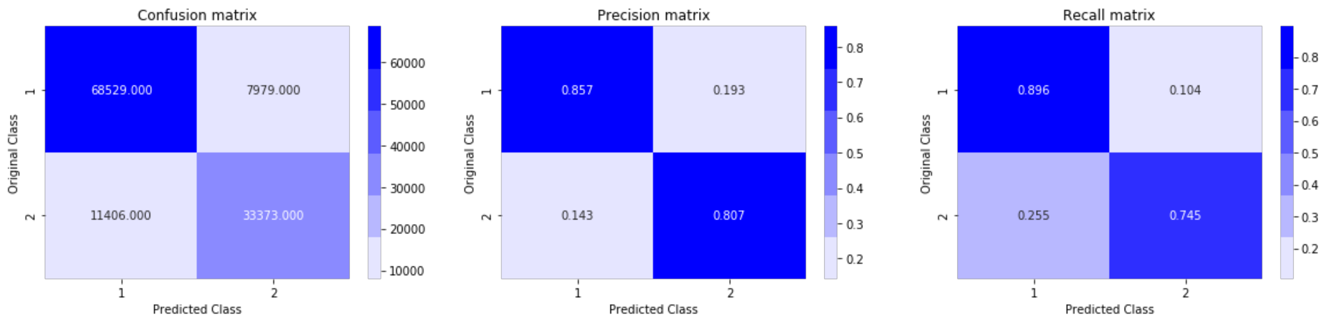
```
100%|████████████████████████████████████████████████████████████████████████████████| 7/7 [09
:24<00:00, 80.61s/it]
```

```
For n_estimators =  500 Train Log Loss  0.32142149225476113 Test Log Loss  0.3297234936906518
```

In [19]:

```
clf=xgb.XGBClassifier(learning_rate=0.1,n_estimators=500,n_jobs=-1)
clf.fit(X_train,y_train)
y_pred=clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, y_pred, eps=1e-15))
predicted_y =np.argmax(y_pred,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

```
The test log loss is: 0.3297234936906518
```



## Hyperparameter tunning using RandomSearch¶

In [23]:

```
from sklearn.model_selection import RandomizedSearchCV
#import tensorflow_addons as tfa

# initialize tqdm callback with default parameters
#tqdm_callback = tfa.callbacks.TQDMProgressBar()
param_grid = {"max_depth":[1,5,10,50,100,500,1000],
              "n_estimators":[50,100,150,200,300,400,500]}

model = RandomizedSearchCV(xgb.XGBClassifier(n_jobs=-1,random_state=25), param_distributions=param_
grid,n_iter=30,scoring='neg_log_loss',cv=3,n_jobs=-1,verbose=10)

model.fit(X_train,y_train)
model.best_params_
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed: 80.3min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 106.0min
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed: 150.8min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 461.0min
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed: 517.4min
[Parallel(n_jobs=-1)]: Done  61 tasks      | elapsed: 913.0min
[Parallel(n_jobs=-1)]: Done  77 out of  90 | elapsed: 1001.1min remaining: 169.0min
[Parallel(n_jobs=-1)]: Done  87 out of  90 | elapsed: 1177.1min remaining: 40.6min
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed: 1217.3min finished
```

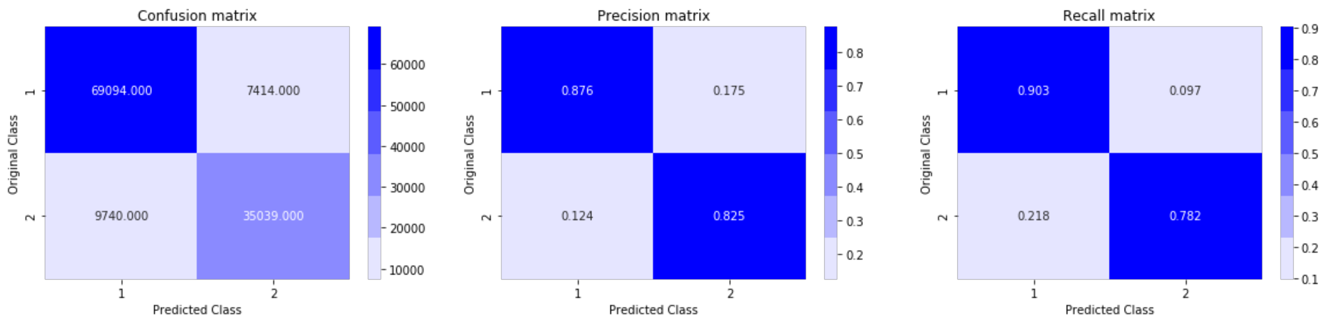
Out[23]:

```
{'n_estimators': 500, 'max_depth': 10}
```

In [24]:

```
clf=xgb.XGBClassifier(n_jobs=-1,random_state=25,max_depth=10,n_estimators=400,verbose=10)
clf.fit(X_train,y_train)
y_pred_test=clf.predict_proba(X_test)
y_pred_train=clf.predict_proba(X_train)
log_loss_train = log_loss(y_train, y_pred_train, eps=1e-15)
log_loss_test=log_loss(y_test,y_pred_test,eps=1e-15)
print('Train log loss = ',log_loss_train,' Test log loss = ',log_loss_test)
predicted_y=np.argmax(y_pred_test,axis=1)
plot_confusion_matrix(y_test,predicted_y)
```

```
Train log loss =  0.2355965479034701  Test log loss =  0.2963621831303785
```



## Procedure and Observation

1) First we did Exploratory Data Analysis on Quora Question Pair data in which we performed such as finding number of different questions, checking for duplicates, number of occurence of questions etc.

2) Then we performed some feature extraction like fuzz ratio, fuzz partial ration, longest common substring etc.

3) After performing feature extraction we applied some visualisation techniques such as pair-plot, violin plot, TSNE etc.

4) Then we peformed tfidf-w2vec vectorizer on pair of questions dataset and then we merged each tfidf-w2vec vectors to our advanced featured vectors.

5) In the next step we applied some machine learning algorithm such as logistic regression, support vector machines etc and found log-loss for both train and test dataset.

6) After choosing best parameters we then plotted confusion matrix, precision matrix and recall matrix for each one.

7) We did same process for tfidf vectorizer at the end of this project.

In [73]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","vectorizer","log loss"]
x.add_row(['Logistic regression','TFIDF w2vec','0.4442784682687898'])
x.add_row(['Linear SVM','TFIDF w2vec','0.4902036861337324'])
x.add_row(['XGBOOST','TFIDF w2vec','0.3554456865287508'])
x.add_row(['Logistic regression','TFIDF ','0.37972555220565674'])
x.add_row(['Linear SVM','TFIDF','0.414170284719538'])
x.add_row(['XGBOOST','TFIDF ','0.2963621831303785'])

print(x)
```

```
+---------------------+-------------+---------------------+
|        Model        |  vectorizer |       log loss      |
+---------------------+-------------+---------------------+
| Logistic regression | TFIDF w2vec |  0.4442784682687898 |
|      Linear SVM     | TFIDF w2vec |  0.4902036861337324 |
|       XGBOOST       | TFIDF w2vec |  0.3554456865287508 |
| Logistic regression |    TFIDF    | 0.37972555220565674 |
|      Linear SVM     |    TFIDF    |  0.414170284719538  |
|       XGBOOST       |    TFIDF    |  0.2963621831303785 |
+---------------------+-------------+---------------------+
```