Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

Problem Statement

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

1.2 Sources/Useful Links

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

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

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in 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 (https://www.kaggle.com/c/quora-question-pairs#evaluation)](https://www.kaggle.com/c/quora-question-pairs#evaluation)

Metric(s):

- log-loss : [https://www.kaggle.com/wiki/LogarithmicLoss (https://www.kaggle.com/wiki/LogarithmicLoss)](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]:
```python
import warnings
warnings.filterwarnings("ignore")

import sys
import os
import gc
import re
import time
import distance
import spacy
import sqlite3
import csv
import math

import datetime as dt
from tqdm import tqdm
from os import path
from PIL import Image

import numpy as np
import pandas as pd
from collections import Counter, defaultdict

import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
from bs4 import BeautifulSoup
from wordcloud import WordCloud, STOPWORDS

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from fuzzywuzzy import fuzz

from sklearn.preprocessing import MinMaxScaler

from sklearn.manifold import TSNE
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
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 sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
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 sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
from mlxtend.classifier import StackingClassifier

from scipy.sparse import hstack

from sqlalchemy import create_engine # database connection

import xgboost as xgb
```

```
C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarning:

Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning
```

3.1 Reading data and basic stats

In [2]:
```python
df=pd.read_csv('train.csv')

print('No of data-points : ',df.shape[0])
```

```
No 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):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

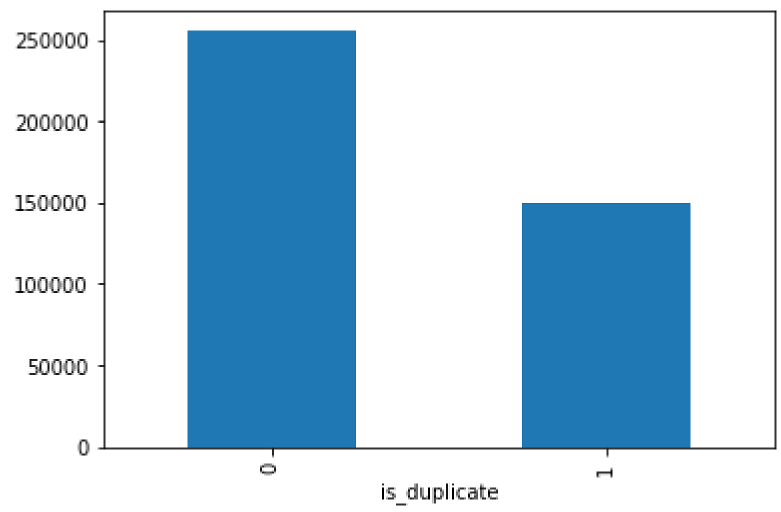
We are given a minimal number of data fields here, consisting of:

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

3.2.1 Distribution of data points among output classes

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

In [5]:
```
df.groupby('is_duplicate')['id'].count().plot.bar()
```
Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d958b8df98>
```



In [6]:
```
print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```
```
~> Total number of question pairs for training:
   404290
```

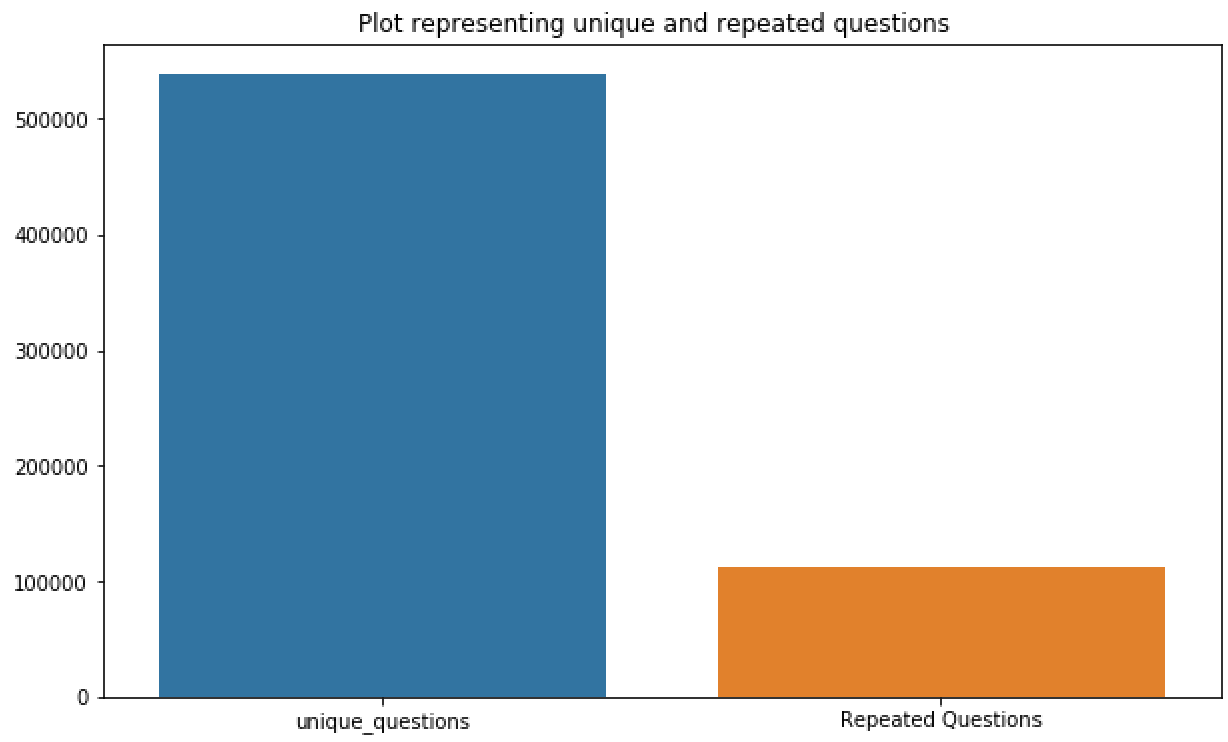
In [7]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round(df['is_duplicate'].mean()*100
, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_duplicate'].mean()*100, 2)))
```
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.2 Number of unique questions

In [8]:
```python
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]:
```python
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

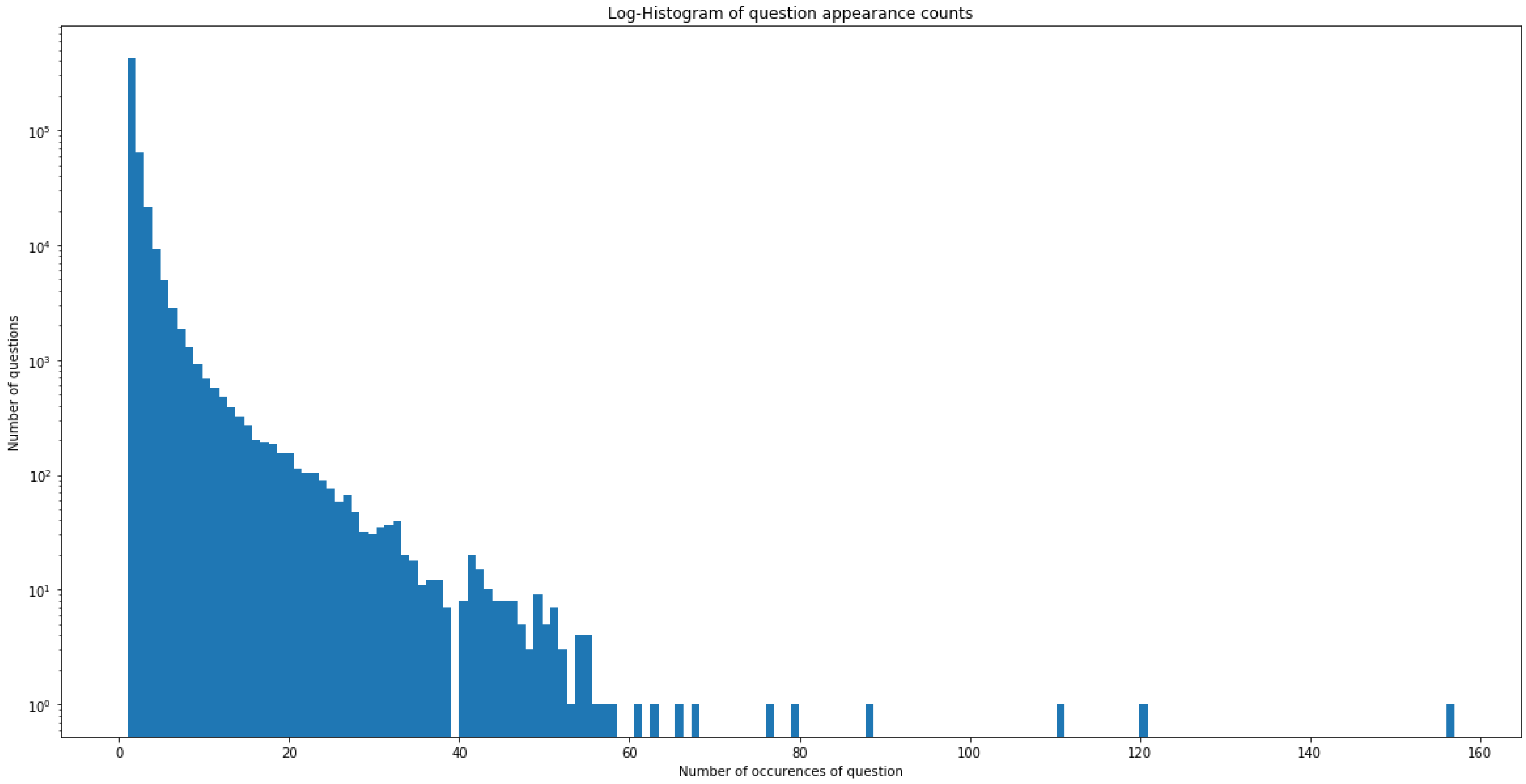
In [10]:
```python
#checking whether there are any repeated pair of questions

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("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('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

    def normalized_word_Common(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * len(w1 & w2)
    df['word_Common'] = df.apply(normalized_word_Common, axis=1)

    def normalized_word_Total(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * (len(w1) + len(w2))
    df['word_Total'] = df.apply(normalized_word_Total, axis=1)

    def normalized_word_share(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
        return 1.0 * len(w1 & w2)/(len(w1) + len(w2))
    df['word_share'] = df.apply(normalized_word_share, axis=1)

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

    df.to_csv("df_fe_without_preprocessing_train.csv", index=False)

df.head()
```

Out[14]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total
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	23.0
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	20.0
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	24.0
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	19.0
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	20.0

3.3.1 Analysis of some of the extracted features

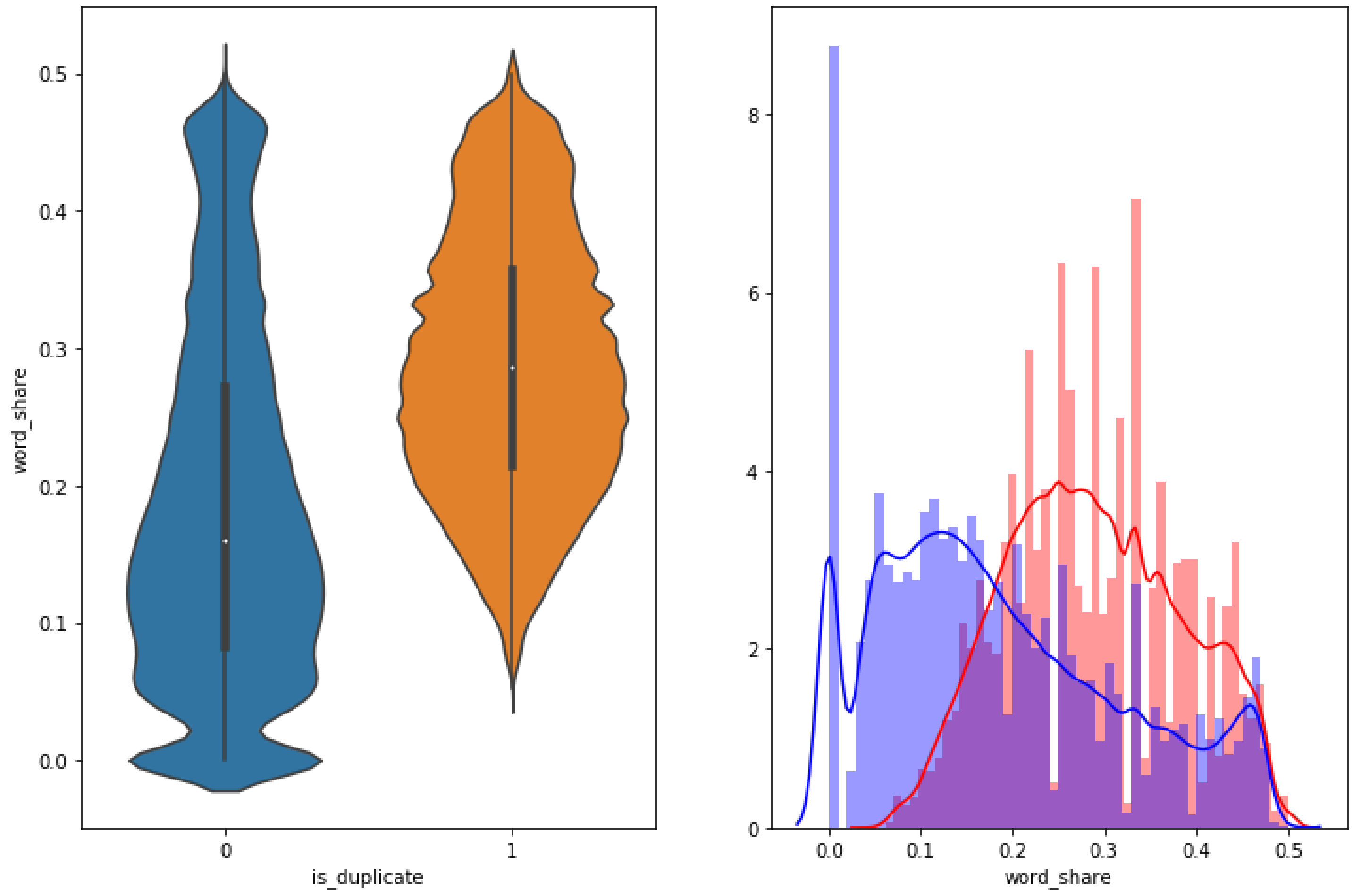
- Here are some questions have only one single words.

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

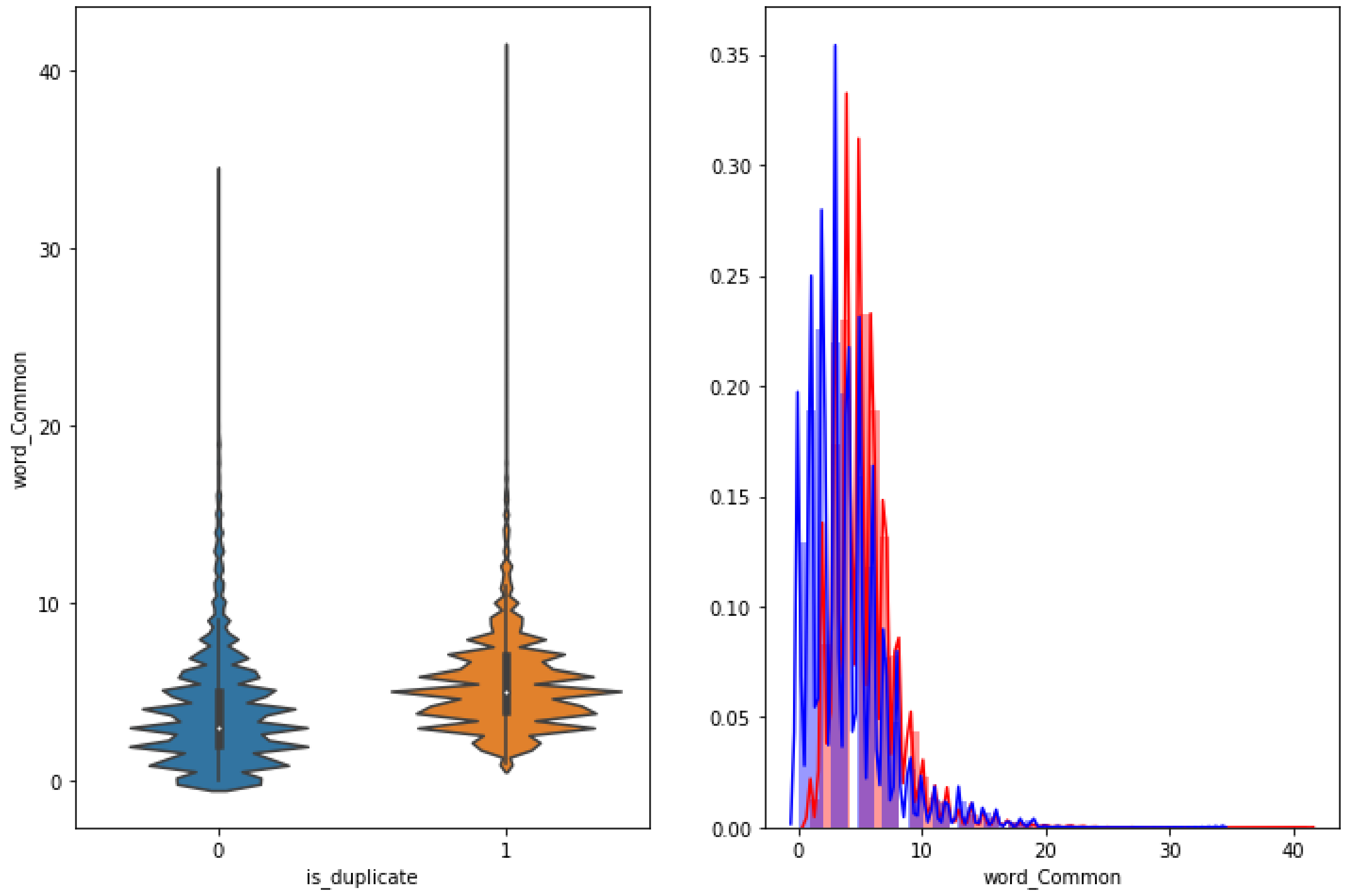
In [16]:
```
plt.figure(figsize=(12, 8))

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()
```



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

In [18]:
```
df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
df = df.fillna('')
df.head()
```

Out[18]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total
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	23.0
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	20.0
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	24.0
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	19.0
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	20.0

3.4 Preprocessing of Text

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

In [19]:
```
# 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't", "can not")\
                           .replace("n't", " not").replace("what's", "what is").replace("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace("'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 (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

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

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

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

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

In [20]:
```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-strings
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"], x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the tokens alphabetically, and
    # then joining them back into a string We then compare the transformed strings with a simple ratio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"], x["question2"]), axis=1)
```

```
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), axis=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"], x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["question1"], x["question2"]), axis=1)
    return df
```

In [21]:
```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head()
```

Out[21]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	first_word_eq	abs_le
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	1.0	
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	1.0	
2	2	5	6	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0	0.399992	0.333328	0.399992	0.249997	...	0.285712	0.0	1.0	
3	3	7	8	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.0	
4	4	9	10	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.666644	...	0.307690	0.0	1.0	

5 rows × 21 columns

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

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

In [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('train_p.txt', p, delimiter=' ', fmt='%s')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s', encoding="utf-8")
```

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

In [24]:
```python
# reading the text files and removing the Stop Words:

d = path.dirname('.')

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

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

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33194892
```

In [25]:
```python
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

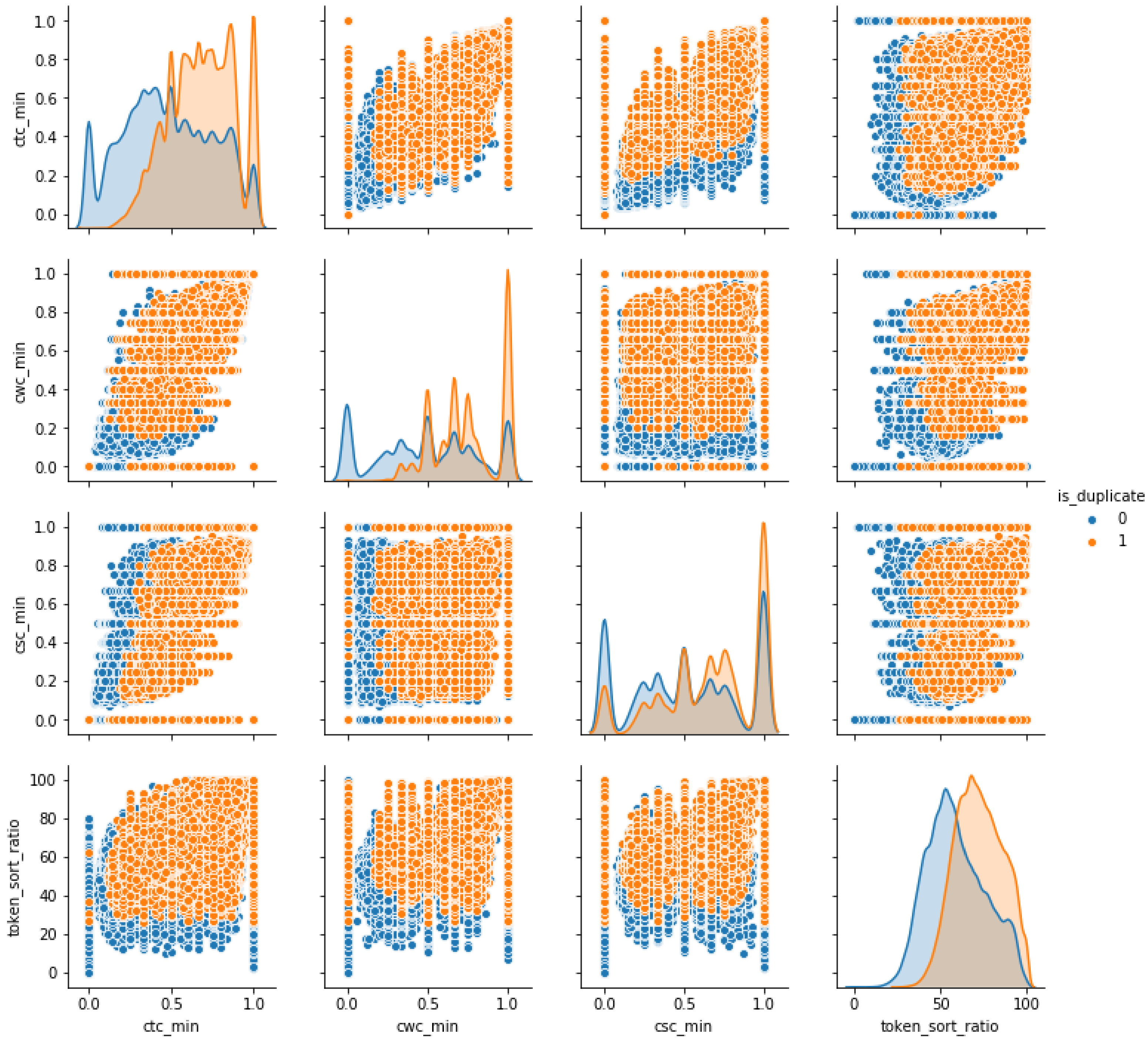
Word Clouds generated from non duplicate pair question's text

In [26]:
```python
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

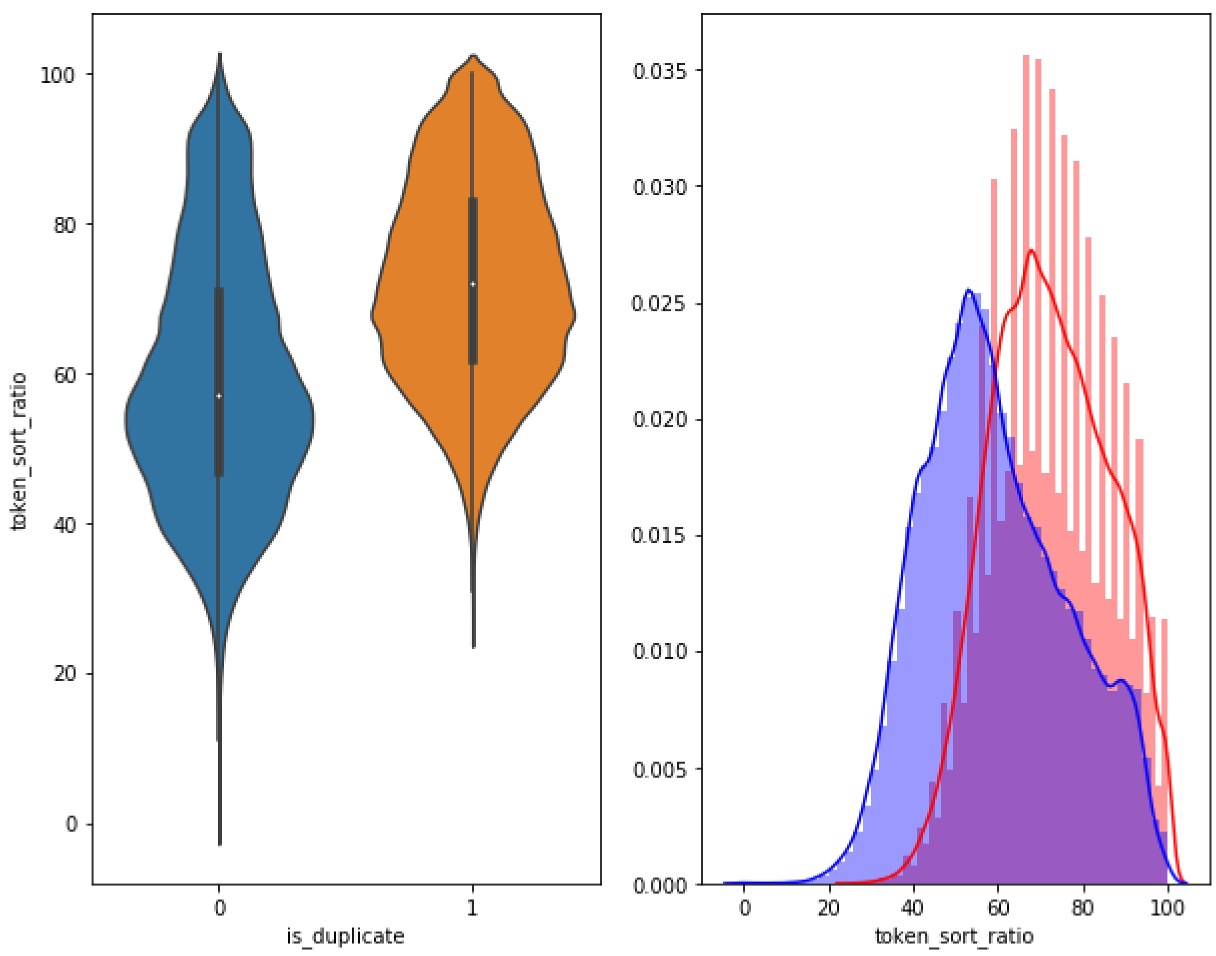
```
Word Cloud for non-Duplicate Question pairs:
```

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

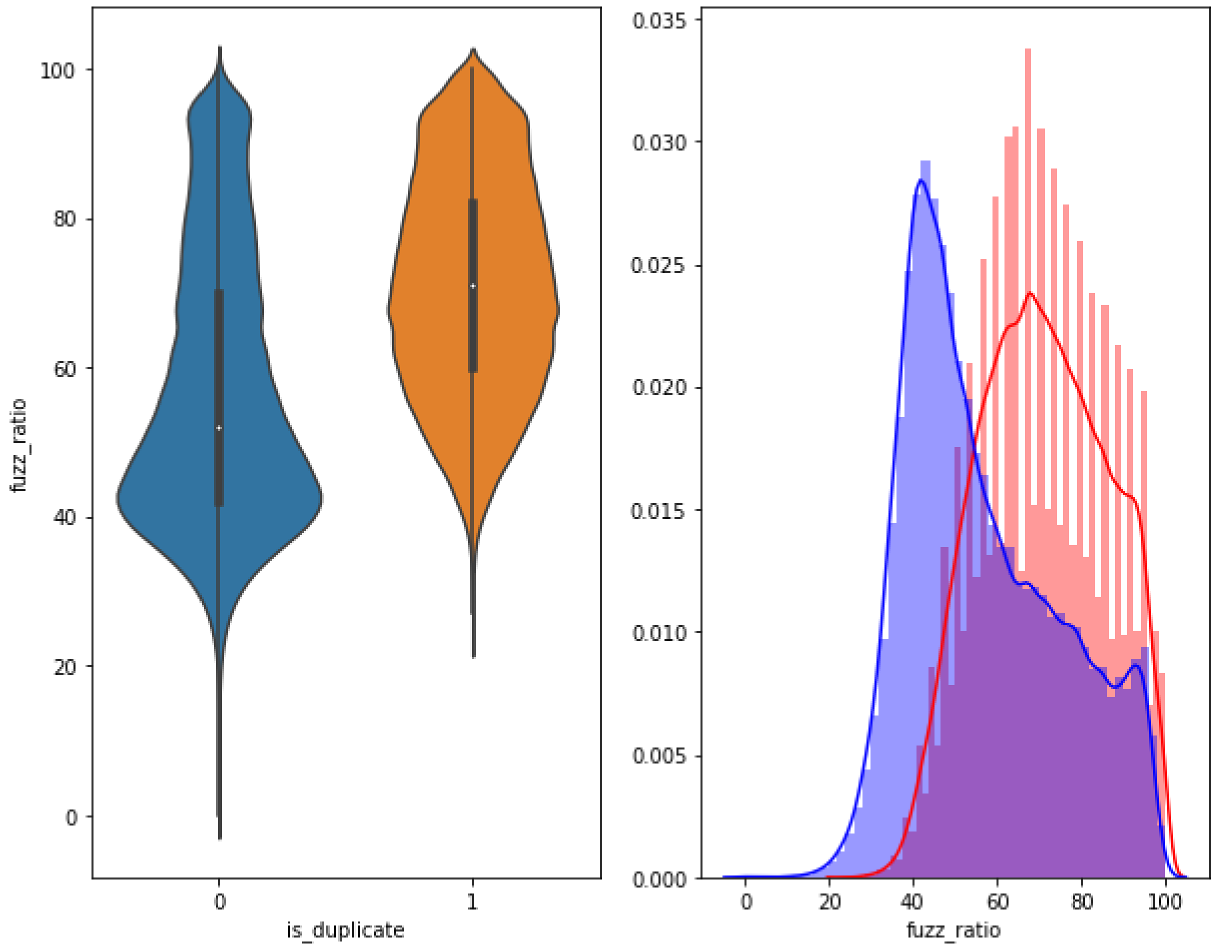
In [27]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=
['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```



In [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()
```



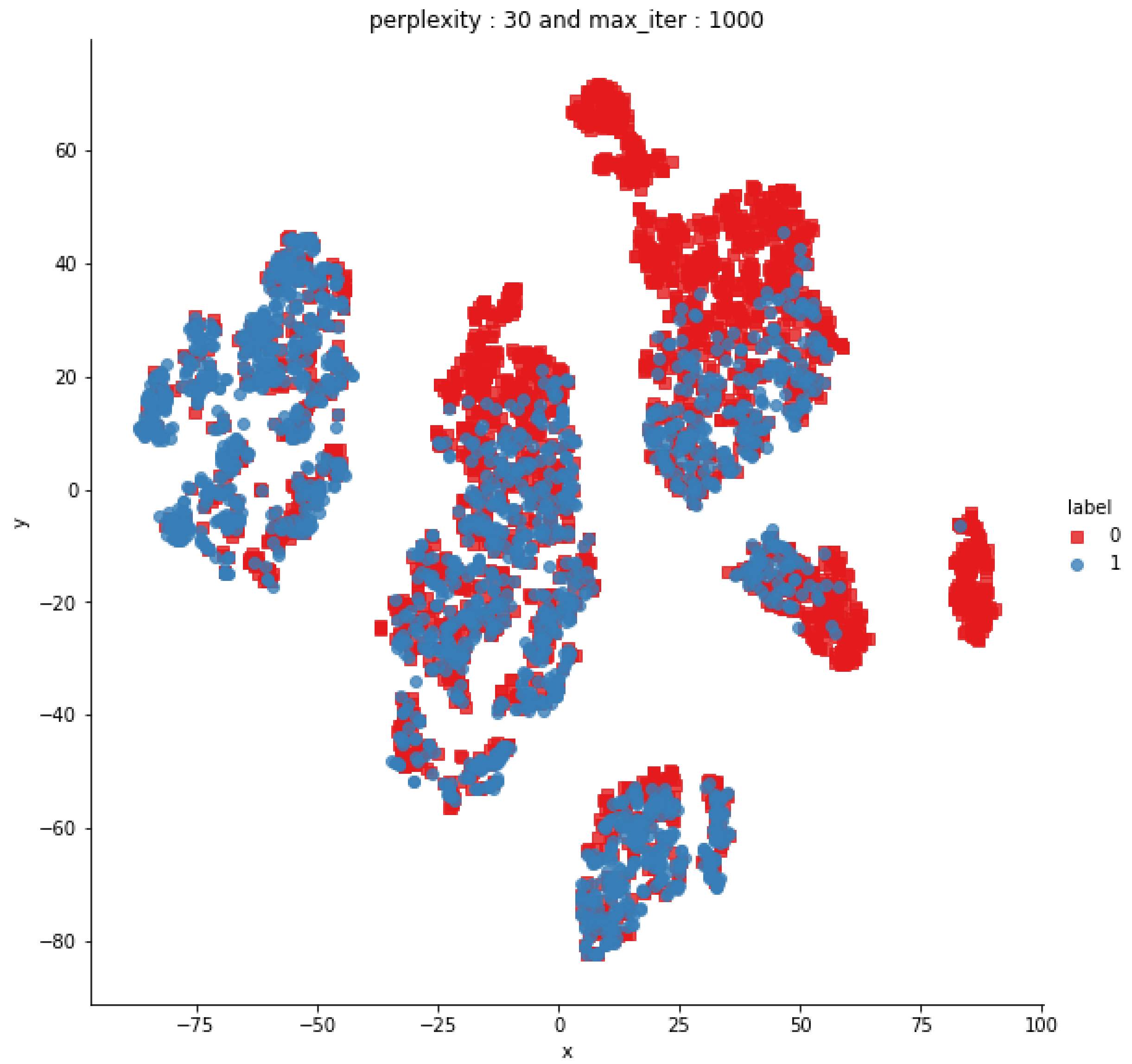
3.5.2 Visualization

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 0.009s...
[t-SNE] Computed neighbors for 5000 samples in 0.373s...
[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 0.278s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iterations in 2.662s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iterations in 1.960s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iterations in 1.969s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iterations in 2.045s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iterations in 2.058s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.273346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 iterations in 2.094s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iterations in 2.018s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iterations in 2.027s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iterations in 2.047s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iterations in 2.042s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iterations in 2.085s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iterations in 2.067s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iterations in 2.063s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iterations in 2.076s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iterations in 2.079s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iterations in 2.065s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iterations in 2.072s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iterations in 2.068s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iterations in 2.080s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iterations in 2.085s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [33]:
```
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, height=8,palette="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```

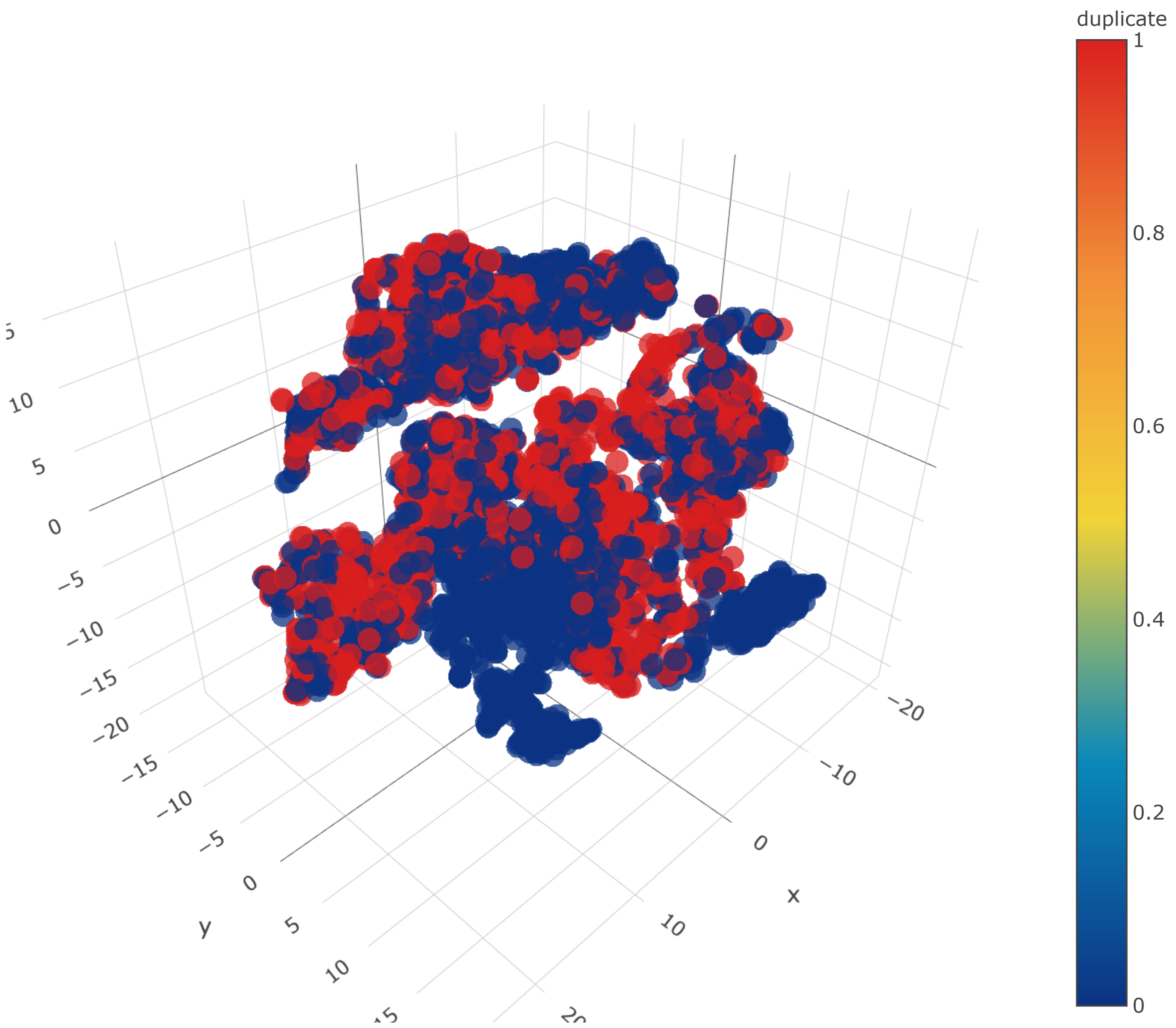


In [34]:
```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.008s...
[t-SNE] Computed neighbors for 5000 samples in 0.405s...
[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 0.332s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 iterations in 10.917s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (50 iterations in 6.338s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (50 iterations in 5.727s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (50 iterations in 5.590s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (50 iterations in 5.551s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729782
[t-SNE] Iteration 300: error = 1.4978341, gradient norm = 0.0006938 (50 iterations in 6.445s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50 iterations in 8.321s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50 iterations in 8.412s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50 iterations in 8.419s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50 iterations in 8.253s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50 iterations in 8.034s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50 iterations in 8.066s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50 iterations in 8.093s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50 iterations in 8.332s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50 iterations in 8.391s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50 iterations in 8.316s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50 iterations in 8.157s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50 iterations in 8.149s)
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50 iterations in 8.138s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (50 iterations in 8.121s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [35]:
```
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')
```

3d embedding with engineered features



3.6 Featurizing text data with tfidf weighted word-vectors

In [36]:
```
# avoid decoding problems
df = pd.read_csv("train.csv")

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

Out[36]:

	Unnamed: 0	id	qid1	qid2	question1	question2	is_duplicate
0	237030	237030	33086	348102	How can I stop playing video games?	Should I stop playing video games with my child?	0
1	247341	247341	73272	8624	Who is better Donald Trump or Hillary Clinton?	Why is Hillary Clinton a better choice than Do...	1
2	246425	246425	359482	359483	What do you think is the chance that sometime ...	Do you think there will be another world war/n...	1
3	306985	306985	1357	47020	Why are so many questions posted to Quora that...	Why do people write questions on Quora that co...	1
4	225863	225863	334315	334316	Can there even be a movie ever rated 10/10 on ...	What are your 10/10 movies?	0

In [37]:
```
dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
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]:
```
df1.head()
```

Out[41]:

	id	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_sc
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	100	
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	86	
2	2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	66	
3	3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	36	
4	4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	67	

In [48]:
```
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 [49]:
```
#removing id from X
X=X.sample(n=100000,random_state=1)
X=X.drop('id',axis=1)
X.columns
```

Out[49]:
```
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 [52]:
```
y=np.array(duplicate.sample(n=100000,random_state=1))
```

In [53]:
```
#splitting data into train and test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=3,test_size=0.3)
```

In [54]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(70000, 27)
(70000,)
(30000, 27)
(30000,)
```

In [55]:
```python
#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 [56]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# list_of_sentance_train=[]
# for sentance in X_train_ques:
#     list_of_sentance_train.append(sentance.split())

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_))
```

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

In [57]:
```python
# 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%|██████████████████████████████████████████| 70000/70000 [15:24<00:00, 76.30it/
s]
```

In [58]:
```python
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%|██████████████████████████████████████████| 30000/30000 [06:38<00:00, 75.21it/
s]
```

In [59]:
```python
first_df=pd.DataFrame(vecs1)
sec_df=pd.DataFrame(vecs2)
```

In [60]:
```
from scipy.sparse import hstack
X_train = hstack((X_train.values,first_df))
X_test= hstack((X_test.values,sec_df))
print(X_train.shape)
print(X_test.shape)
```

```
(70000, 122)
(30000, 122)
```

4. Machine Learning Models

In [61]:
```
print("Number of data points in train data :",X_train.shape)
print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (70000, 122)
Number of data points in test data : (30000, 122)
```

In [62]:
```
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)
```

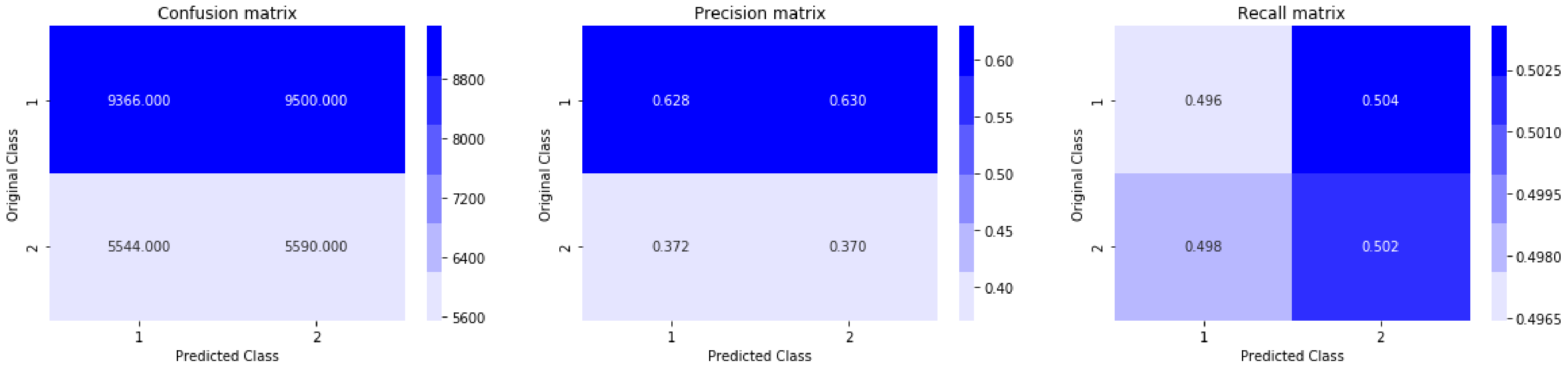
```
---------- Distribution of output variable in train data ----------
Class 0:  0.6313714285714286 Class 1:  0.3686285714285714
---------- Distribution of output variable in train data ----------
Class 0:  0.3711333333333333 Class 1:  0.3711333333333333
```

In [63]:
```python
# 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 [64]:
```
# 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)
```

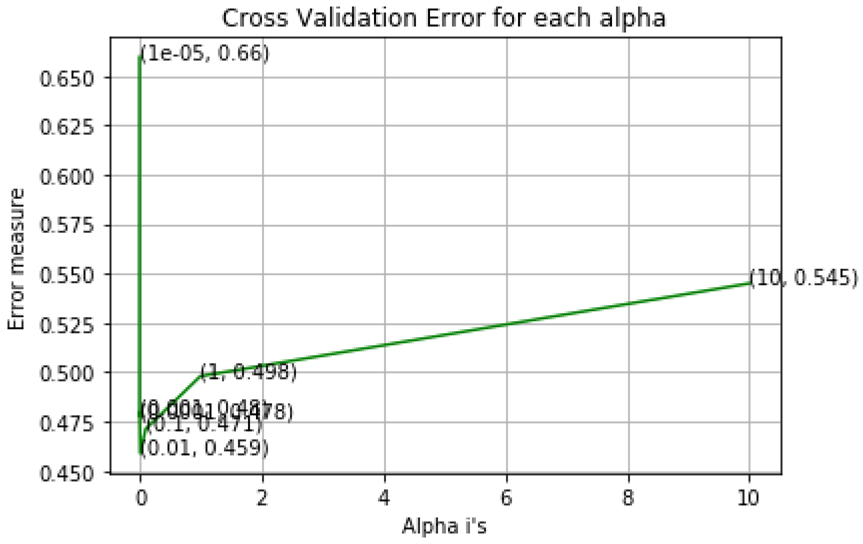
```
Log loss on Test Data using Random Model 0.8809623427268886
```



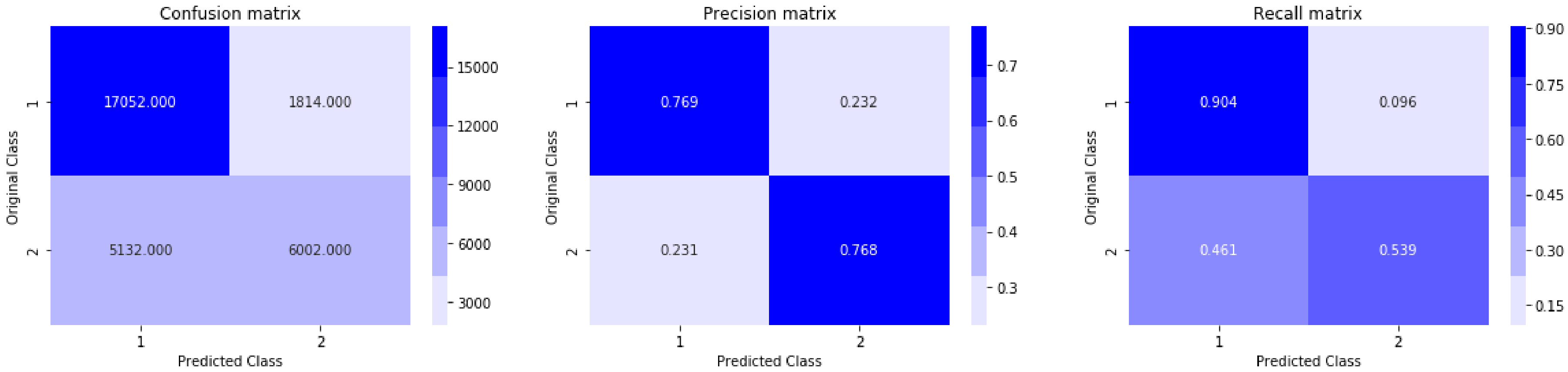
4.4 Logistic Regression with hyperparameter tuning

In [65]:
```python
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.SGDClassifi
er.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='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 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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', 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=c
lf.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)
```

```
For values of alpha =  1e-05 The log loss is: 0.659569450346315
For values of alpha =  0.0001 The log loss is: 0.4779586564734812
For values of alpha =  0.001 The log loss is: 0.4796557470998263
For values of alpha =  0.01 The log loss is: 0.4589325875931541
For values of alpha =  0.1 The log loss is: 0.4712408524606177
For values of alpha =  1 The log loss is: 0.49800720247415936
For values of alpha =  10 The log loss is: 0.5450365811903767
```




```
For values of best alpha =  0.01 The train log loss is: 0.45784126269525527
For values of best alpha =  0.01 The test log loss is: 0.4589325875931541
Total number of data points : 30000
```

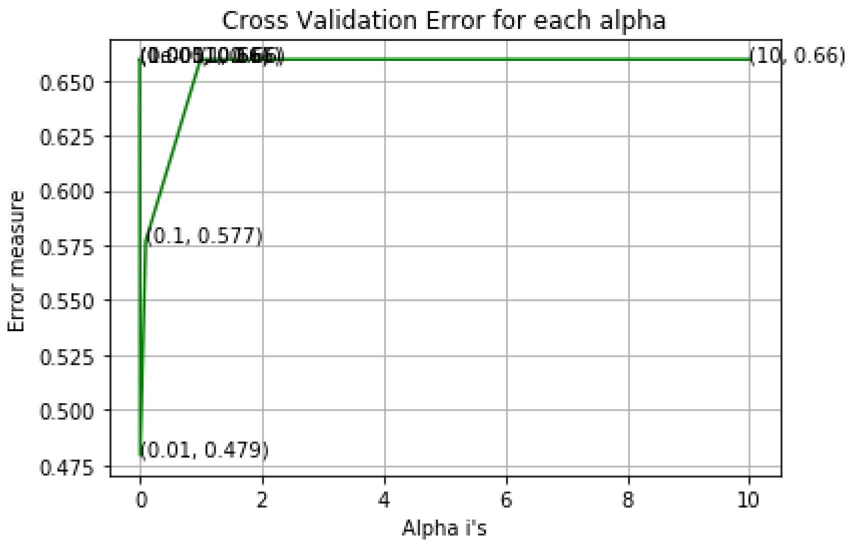


4.5 Linear SVM with hyperparameter tuning

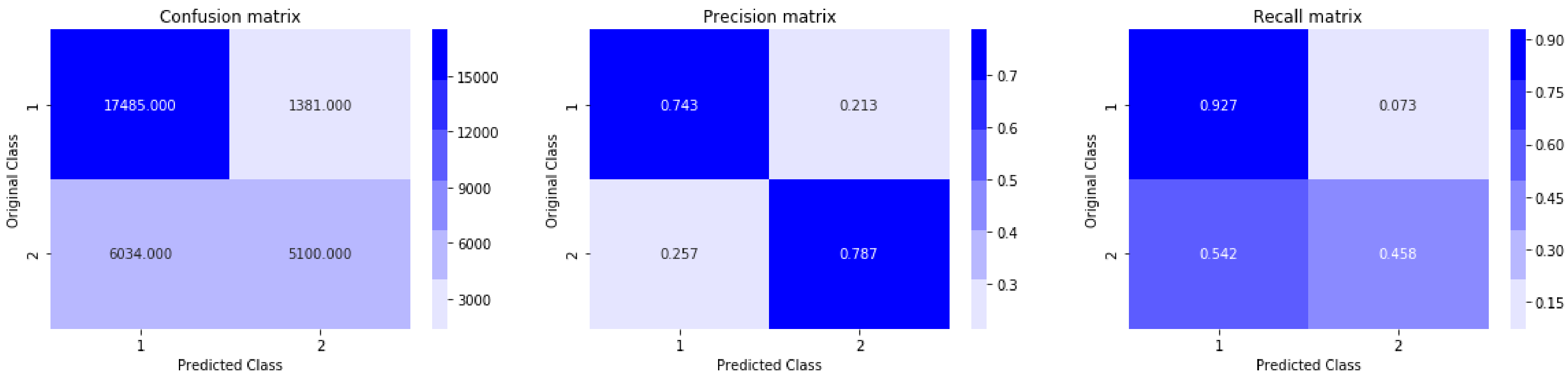
In [68]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifi
er.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='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 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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='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=c
lf.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)
```

```
For values of alpha =  1e-05 The log loss is: 0.659569450346315
For values of alpha =  0.0001 The log loss is: 0.659569450346315
For values of alpha =  0.001 The log loss is: 0.659569450346315

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of alpha =  0.01 The log loss is: 0.47945018941308165

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of alpha =  0.1 The log loss is: 0.5770116924288787

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of alpha =  1 The log loss is: 0.659569450346315

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.
```

```
For values of alpha =  10 The log loss is: 0.659569450346315
```




```
C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of best alpha =  0.01 The train log loss is: 0.4810816426639049
For values of best alpha =  0.01 The test log loss is: 0.47945018941308165
Total number of data points : 30000
```



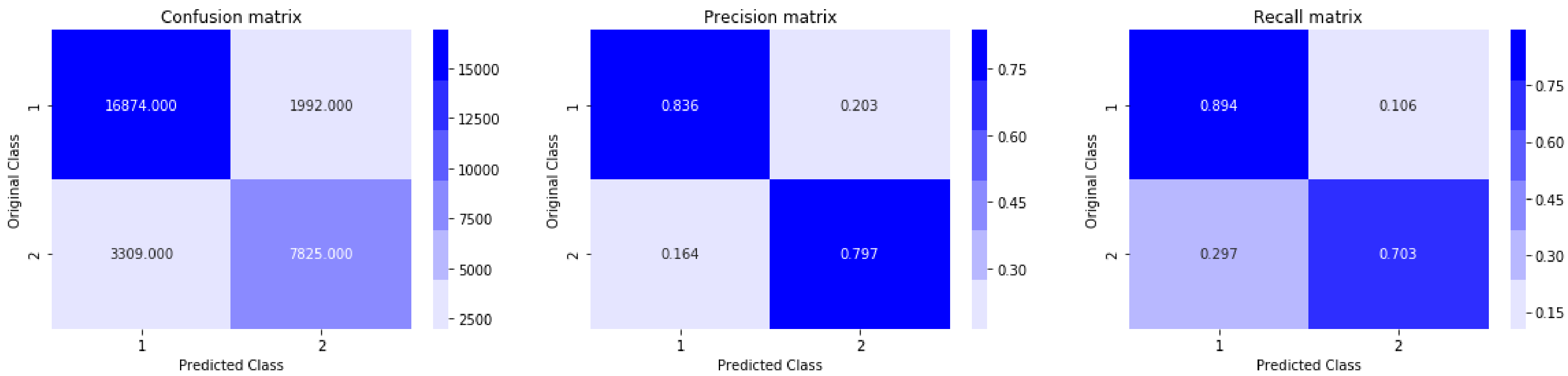
4.6 XGBoost

In [69]:
```
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:0.684783	valid-logloss:0.684785
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:0.615379	valid-logloss:0.615332
[20]	train-logloss:0.563996	valid-logloss:0.564006
[30]	train-logloss:0.525557	valid-logloss:0.52545
[40]	train-logloss:0.496196	valid-logloss:0.495999
[50]	train-logloss:0.473053	valid-logloss:0.472965
[60]	train-logloss:0.454687	valid-logloss:0.454648
[70]	train-logloss:0.439869	valid-logloss:0.439866
[80]	train-logloss:0.427752	valid-logloss:0.427832
[90]	train-logloss:0.418106	valid-logloss:0.418294
[100]	train-logloss:0.410097	valid-logloss:0.41035
[110]	train-logloss:0.403362	valid-logloss:0.403704
[120]	train-logloss:0.397634	valid-logloss:0.39801
[130]	train-logloss:0.392905	valid-logloss:0.393443
[140]	train-logloss:0.388966	valid-logloss:0.389659
[150]	train-logloss:0.385655	valid-logloss:0.386527
[160]	train-logloss:0.382422	valid-logloss:0.383495
[170]	train-logloss:0.379591	valid-logloss:0.380849
[180]	train-logloss:0.377024	valid-logloss:0.378474
[190]	train-logloss:0.374732	valid-logloss:0.376362
[200]	train-logloss:0.372414	valid-logloss:0.374236
[210]	train-logloss:0.370519	valid-logloss:0.372551
[220]	train-logloss:0.368624	valid-logloss:0.370923
[230]	train-logloss:0.366938	valid-logloss:0.369463
[240]	train-logloss:0.36508	valid-logloss:0.36783
[250]	train-logloss:0.363363	valid-logloss:0.366325
[260]	train-logloss:0.361757	valid-logloss:0.364973
[270]	train-logloss:0.360357	valid-logloss:0.363814
[280]	train-logloss:0.358748	valid-logloss:0.362442
[290]	train-logloss:0.357388	valid-logloss:0.361318
[300]	train-logloss:0.355997	valid-logloss:0.36016
[310]	train-logloss:0.354725	valid-logloss:0.359176
[320]	train-logloss:0.353584	valid-logloss:0.358293
[330]	train-logloss:0.352436	valid-logloss:0.357364
[340]	train-logloss:0.351338	valid-logloss:0.356534
[350]	train-logloss:0.35027	valid-logloss:0.355686
[360]	train-logloss:0.34926	valid-logloss:0.354916
[370]	train-logloss:0.348287	valid-logloss:0.354176
[380]	train-logloss:0.34739	valid-logloss:0.353559
[390]	train-logloss:0.346493	valid-logloss:0.352869
[399]	train-logloss:0.345765	valid-logloss:0.352324
The test log loss is: 0.35232425329248845
```

In [70]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

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



For TFIDF features

In [71]:
```
dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
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 [79]:
```
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 [80]:
```
#removing id from X
X=X.sample(n=100000,random_state=1)
X=X.drop('id',axis=1)
X.columns
```

Out[80]:
```
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 [81]:
```
y=np.array(duplicate.sample(n=100000, random_state=1))
```

In [82]:
```
#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 [83]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(70000, 27)
(70000,)
(30000, 27)
(30000,)
```

In [84]:
```
#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 [85]:
```
#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 [86]:
```
#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)
```

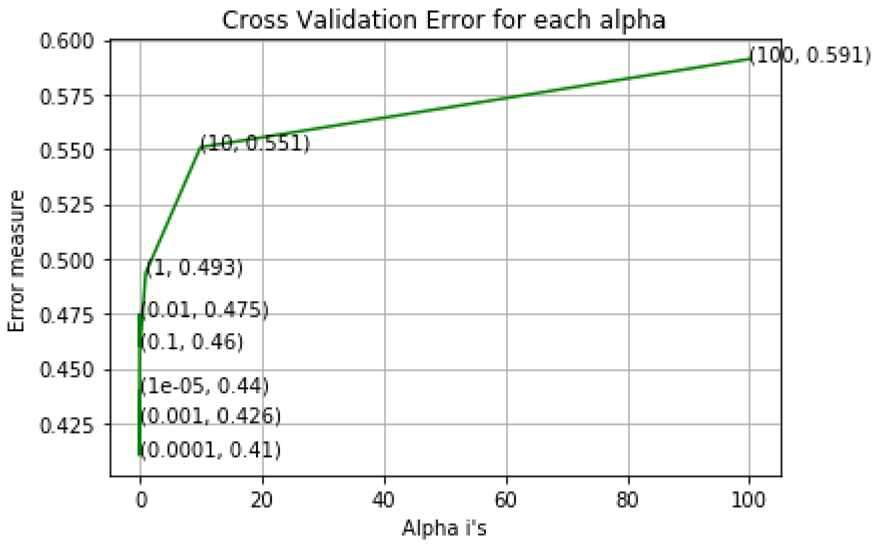
```
(70000, 29456)
(30000, 29456)
```

In [0]:
```
# #standardising data
# from sklearn import preprocessing
# scaler = preprocessing.StandardScaler(with_mean=False)
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)
```

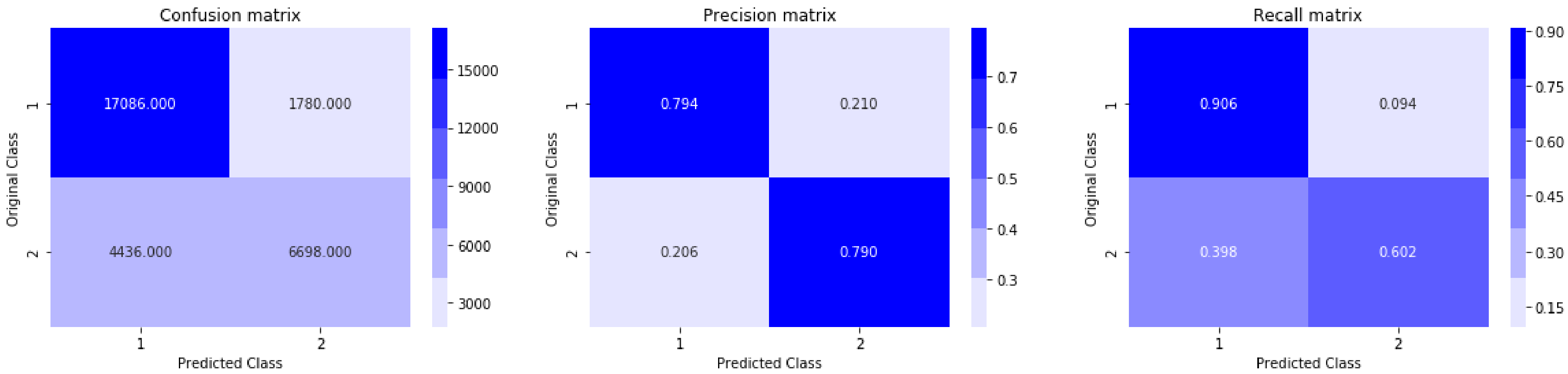
Applying Logistic Regression

In [87]:
```python
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.SGDClassifi
er.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='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 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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', 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=c
lf.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)
```

```
For values of alpha =  1e-05 The log loss is: 0.4397794941988581
For values of alpha =  0.0001 The log loss is: 0.41038385102902347
For values of alpha =  0.001 The log loss is: 0.42618780519923977
For values of alpha =  0.01 The log loss is: 0.4749343797517387
For values of alpha =  0.1 The log loss is: 0.460284472420409
For values of alpha =  1 The log loss is: 0.4934323792611642
For values of alpha =  10 The log loss is: 0.5509389887450498
For values of alpha =  100 The log loss is: 0.591230777370631
```




```
For values of best alpha =  0.0001 The train log loss is: 0.40905566954009797
For values of best alpha =  0.0001 The test log loss is: 0.41038385102902347
Total number of data points : 30000
```

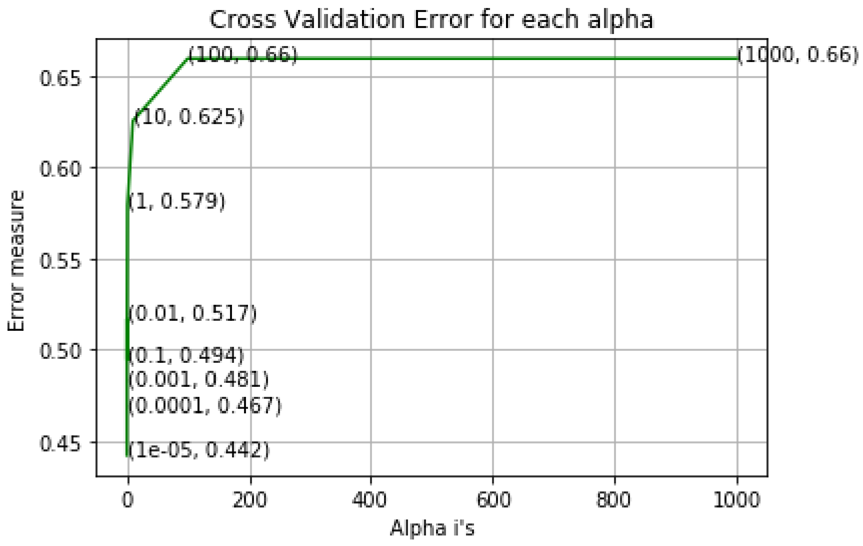


Applying Linear SVM

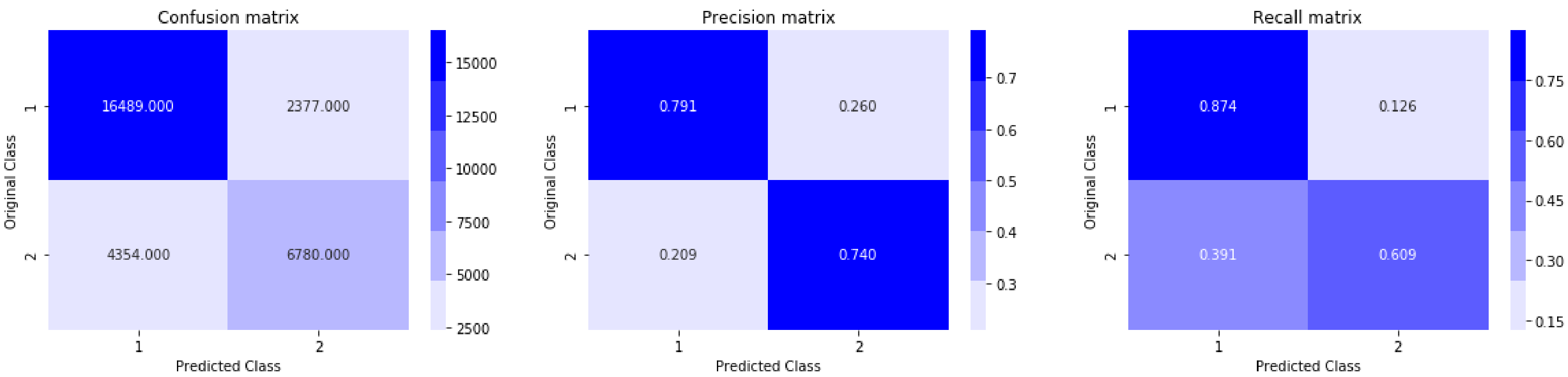
In [95]:

```
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.SGDClassifi
er.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='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 alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', max_iter=2000,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.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='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=c
lf.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)
```

```
For values of alpha =  1e-05 The log loss is: 0.4421283559766547
For values of alpha =  0.0001 The log loss is: 0.46679390109932795
For values of alpha =  0.001 The log loss is: 0.48078236299856586

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of alpha =  0.01 The log loss is: 0.5169389559547751
For values of alpha =  0.1 The log loss is: 0.4942528201101975

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of alpha =  1 The log loss is: 0.5793390149562515

C:\Users\Chandrashekhar\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:561: ConvergenceWarni
ng:

Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.


For values of alpha =  10 The log loss is: 0.6253433082120667
For values of alpha =  100 The log loss is: 0.6595695712025925
For values of alpha =  1000 The log loss is: 0.6595695712025925
```




```
For values of best alpha =  1e-05 The train log loss is: 0.43867285303575293
For values of best alpha =  1e-05 The test log loss is: 0.4421283559766547
Total number of data points : 30000
```



XGBOOST

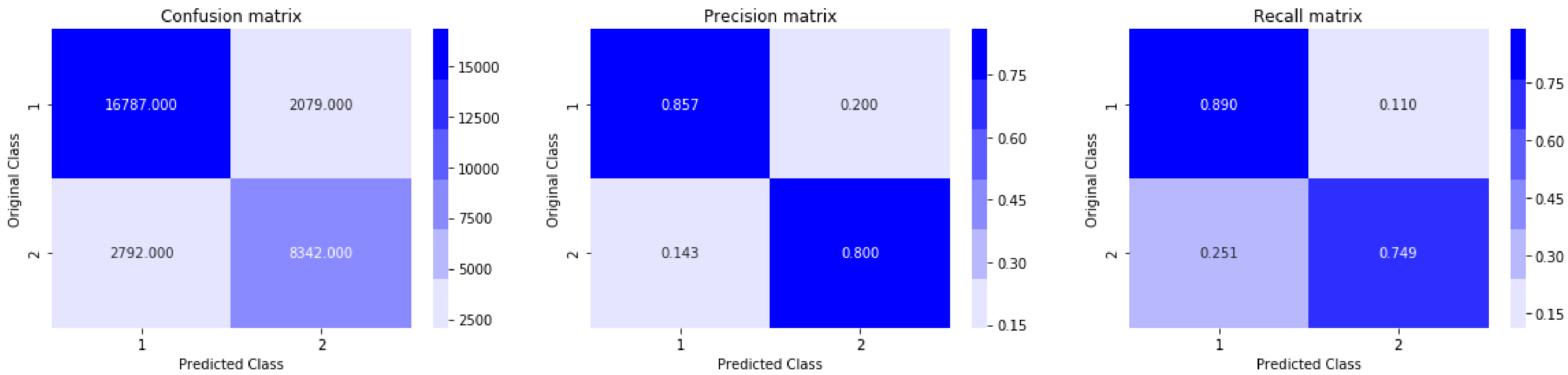
In [89]:
```
import xgboost as xgb
```

In [90]:
```
n_estimators = [50,100,150,200,300,400,500]
test_scores = []
train_scores = []
for i in 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)
```

```
For n_estimators =  50 Train Log Loss  0.37898192255261487 Test Log Loss  0.37860485697182206
For n_estimators =  100 Train Log Loss  0.35815231763182925 Test Log Loss  0.36003871897655576
For n_estimators =  150 Train Log Loss  0.34699334960919137 Test Log Loss  0.35129139623123484
For n_estimators =  200 Train Log Loss  0.33864828361066507 Test Log Loss  0.3453583996747029
For n_estimators =  300 Train Log Loss  0.3272454889531553 Test Log Loss  0.33813053422864503
For n_estimators =  400 Train Log Loss  0.3189324778813943 Test Log Loss  0.33394499660083552
For n_estimators =  500 Train Log Loss  0.3127764263589066 Test Log Loss  0.33172914778997 47
```

In [91]:
```
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: 0.3317291477899747
```



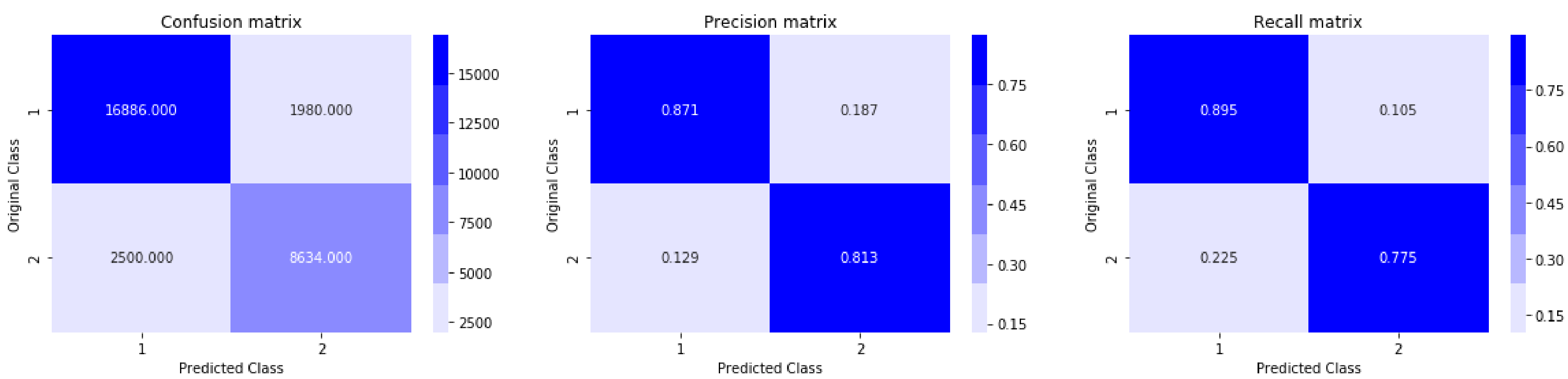
Hyperparameter tunning using RandomSearch

In [93]:
```
from sklearn.model_selection import RandomizedSearchCV
param_grid = {"max_depth":[1,5,10,15,20],
              "n_estimators":[50,100,200,300,400,500]}

model = RandomizedSearchCV(xgb.XGBClassifier(n_jobs=-1,random_state=25), param_distributions=param_grid,n_iter=30,scor
ing='neg_log_loss',cv=3,n_jobs=-1)

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

Out[93]:
```
{'n_estimators': 500, 'max_depth': 10}
```

In [94]:
```
clf=xgb.XGBClassifier(n_jobs=-1,random_state=25,max_depth=10,n_estimators=500)
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 =  0.18796567780874476  Test log loss =  0.30968711726649073
```



Procedure and Observation

In [96]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","vectorizer","log loss"]
x.add_row(['Logistic regression','TFIDF w2vec','0.4589'])
x.add_row(['Linear SVM','TFIDF w2vec','0.4794'])
x.add_row(['XGBOOST','TFIDF w2vec','0.3523'])
x.add_row(['Logistic regression','TFIDF ','0.4103'])
x.add_row(['Linear SVM','TFIDF','0.4421'])
x.add_row(['XGBOOST','TFIDF ','0.3096'])

print(x)
```

```
+---------------------+-------------+----------+
|        Model        |  vectorizer | log loss |
+---------------------+-------------+----------+
| Logistic regression | TFIDF w2vec |  0.4589  |
|      Linear SVM     | TFIDF w2vec |  0.4794  |
|       XGBOOST       | TFIDF w2vec |  0.3523  |
| Logistic regression |    TFIDF    |  0.4103  |
|      Linear SVM     |    TFIDF    |  0.4421  |
|       XGBOOST       |    TFIDF    |  0.3096  |
+---------------------+-------------+----------+
```

Step By Step Process of Model Implementation

Tokenizer: TFIDF Weighted W2V

1. First we have applied simple Random Model(Dumb Model), which gives the log loss of 0.88, that means, the other models has to produce less than 0.88.
2. After that we have applied Logistic Regression on ~100K dataset with hyperparameter tuning, which producs the log loss of 0.458, which is significantly lower than Random Model.
3. We have applied Linear SVM on ~100K dataset with hyperparameter tuning, which produces the log loss of 0.479, which is slightly higher than Logistic Regression.
4. We applied XGBoost Model on ~100k dataset with no hyperparameter tuning, which produces the log loss of 0.35, which is significantly lower than Linear SVM.

As we know that, on high dimension dataset 'XGBoost' does not perform well, but it does perform well in above dataset because of low dimension of 122.Whereas 'Logistic Regression' and 'Linear SVM' performs moderately on low dimension data.

Tokenizer: TFIDF

1. We have applied Logistic Regression on ~100K dataset (performed using TFIDF) with hyperparameter tuning, which produces the log loss of 0.4103, which is significantly lower than previous logistic regression model(performed using TFIDF Weighted W2V).
2. We have applied Linear SVM on ~100K dataset (performed using TFIDF) with hyperparameter tuning, which produces the log loss of 0.4421, which is slightly higher than Logistic Regression, but it is lower than previous Linear SVM model(performed using TFIDF Weighted W2V).
3. We applied XGBoost Model on ~100k dataset (performed using TFIDF) with hyperparameter tuning, which produces the log loss of 0.3096, which is significantly lower than Linear SVM.

Finally for this case study, we conclude that on low dimesion data,we will use hyperparameter tuned 'XGBoost' model and for high dimension data we will use either 'Linear SVM' or 'Logistic Regression'

In [0]: