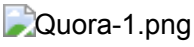


Quora Question Pairs

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?","W
hat 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 happe
n 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 geol
ogist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my
Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
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
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
import warnings
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from 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
```

```
C:\Users\admin\Anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarnin
g:

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

3.1 Reading data and basic stats

In [2]:
```
df = pd.read_csv("C:\\Users\\admin\\Downloads\\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):
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:

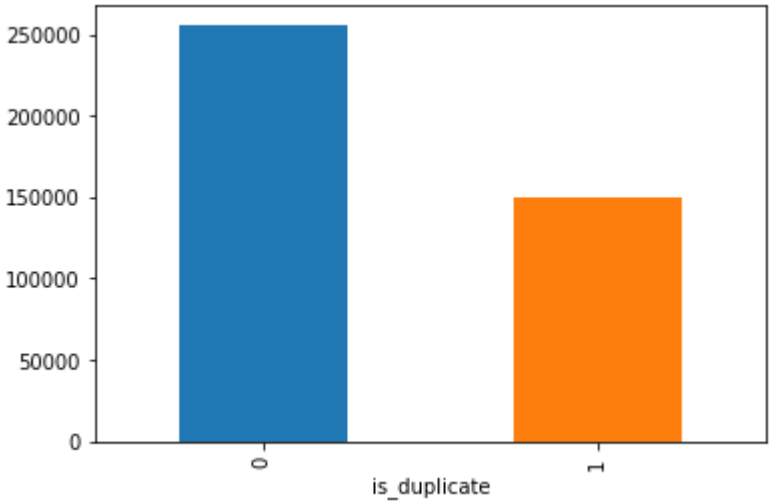
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

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 0xbf4e037a58>



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(1
00 - round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(rou
nd(df['is_duplicate'].mean()*100, 2)))
```

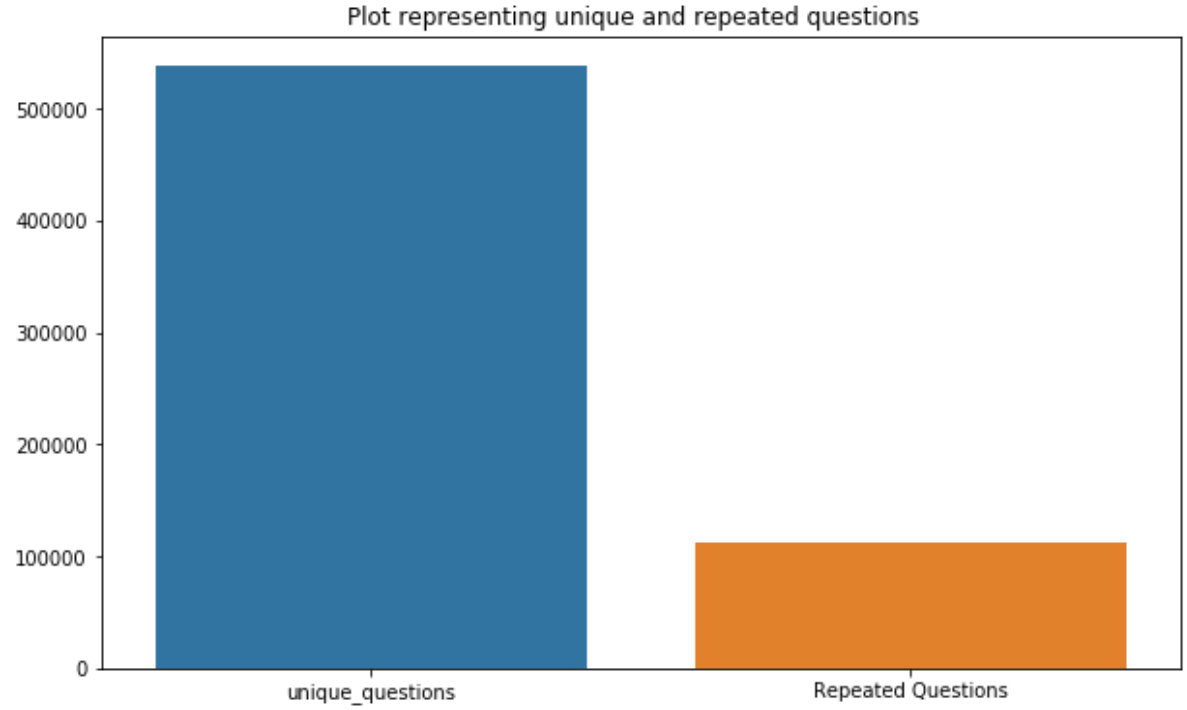
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

~> 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(qi
ds.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 (20.7795394
5937505%)

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

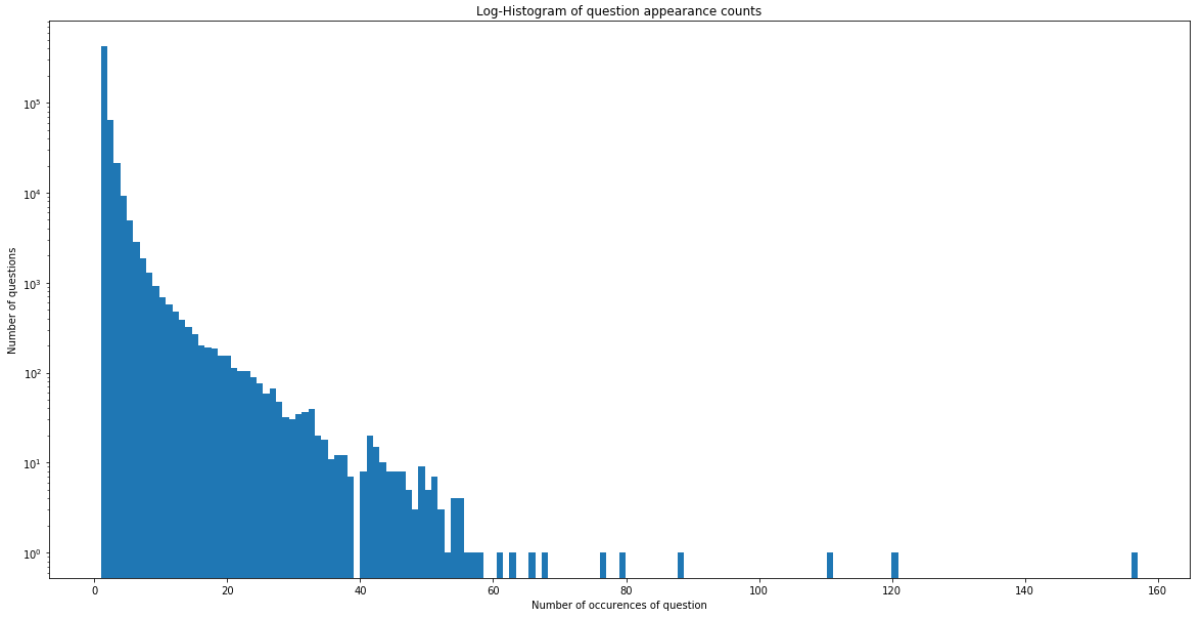
In [10]:
```
#checking whether there are any repeated pair of questions

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(ma
x(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)
```

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

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

In [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	q2le
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39

3.3.1 Analysis of some of the extracted features

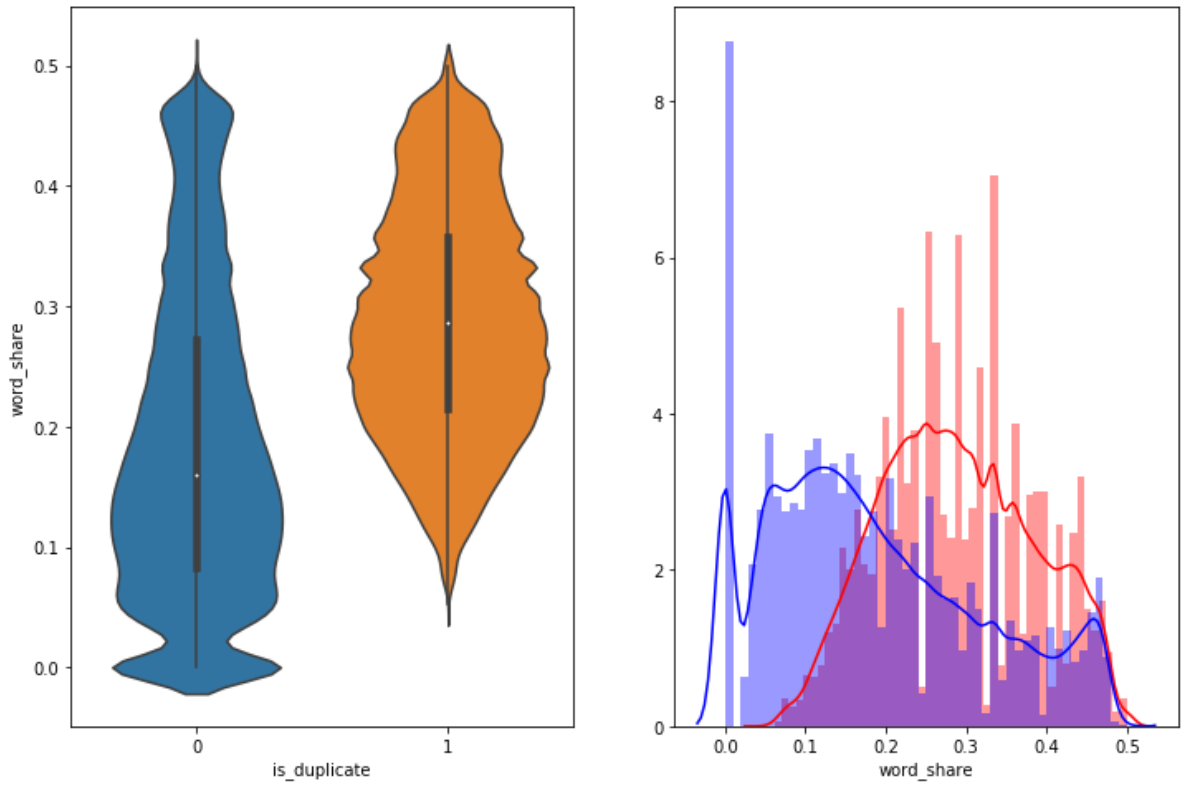
- Here are some questions have only one single words.

In [82]:
```
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_wo
rds']== 1].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_wo
rds']== 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

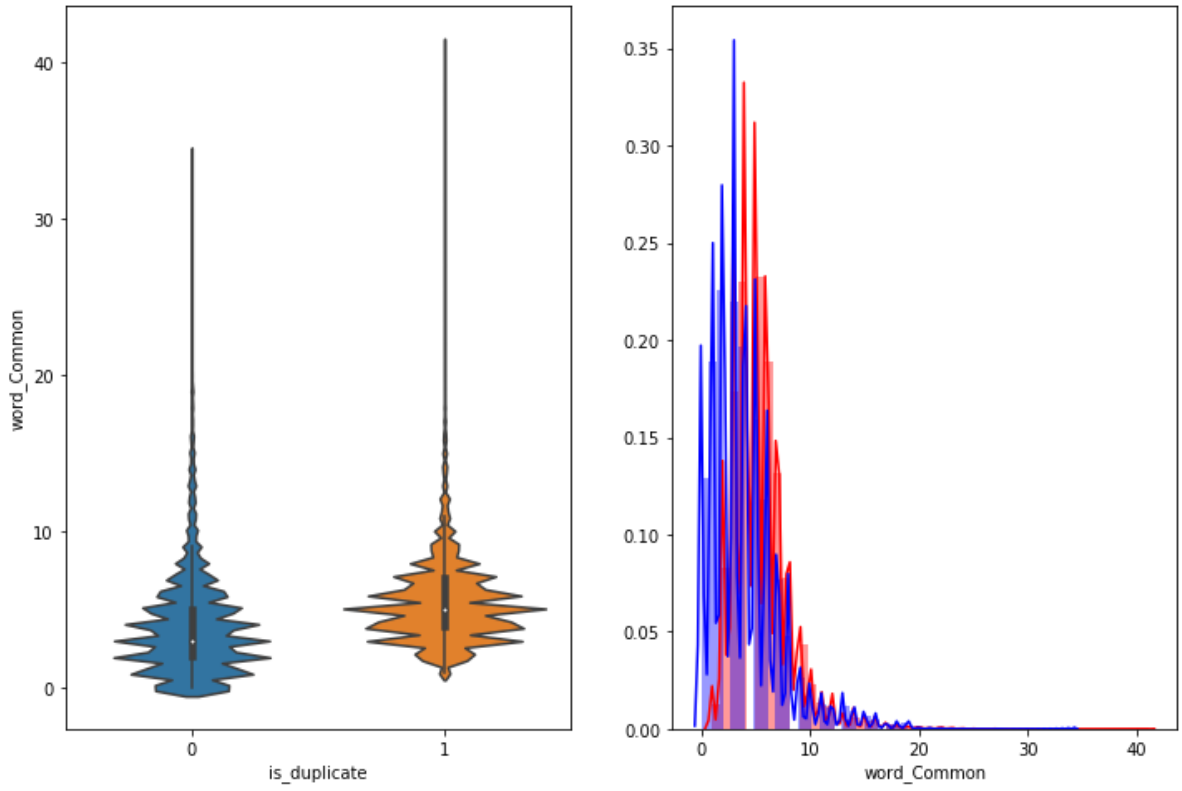
In [83]:
```
plt.figure(figsize=(12, 8))

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", co
lor = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , c
olor = '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 [84]:
```
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", c
olor = '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 [85]:
```
if os.path.isfile('C:\\Users\\admin\\Downloads\\df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("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 [86]:
```
df.head(2)
```

Out[86]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88

In [87]:
```
# 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("’", "'").r
eplace("’", "'")\
                           .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
```

In [88]:
```
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_word
s)) + SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_word
s)) + SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stop
s)) + SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stop
s)) + SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_to
kens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_to
kens)) + 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-str
ing-matching-in-python/
    # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-f
unction-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, s
orting 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["questio
n1"], 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_rat
io(x["question1"], x["question2"]), axis=1)
    return df
```

In [89]:
```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[89]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_
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	0.99
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	0.59

2 rows × 21 columns

In [90]:
```
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}} t
o {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 [91]:
```
# 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
```

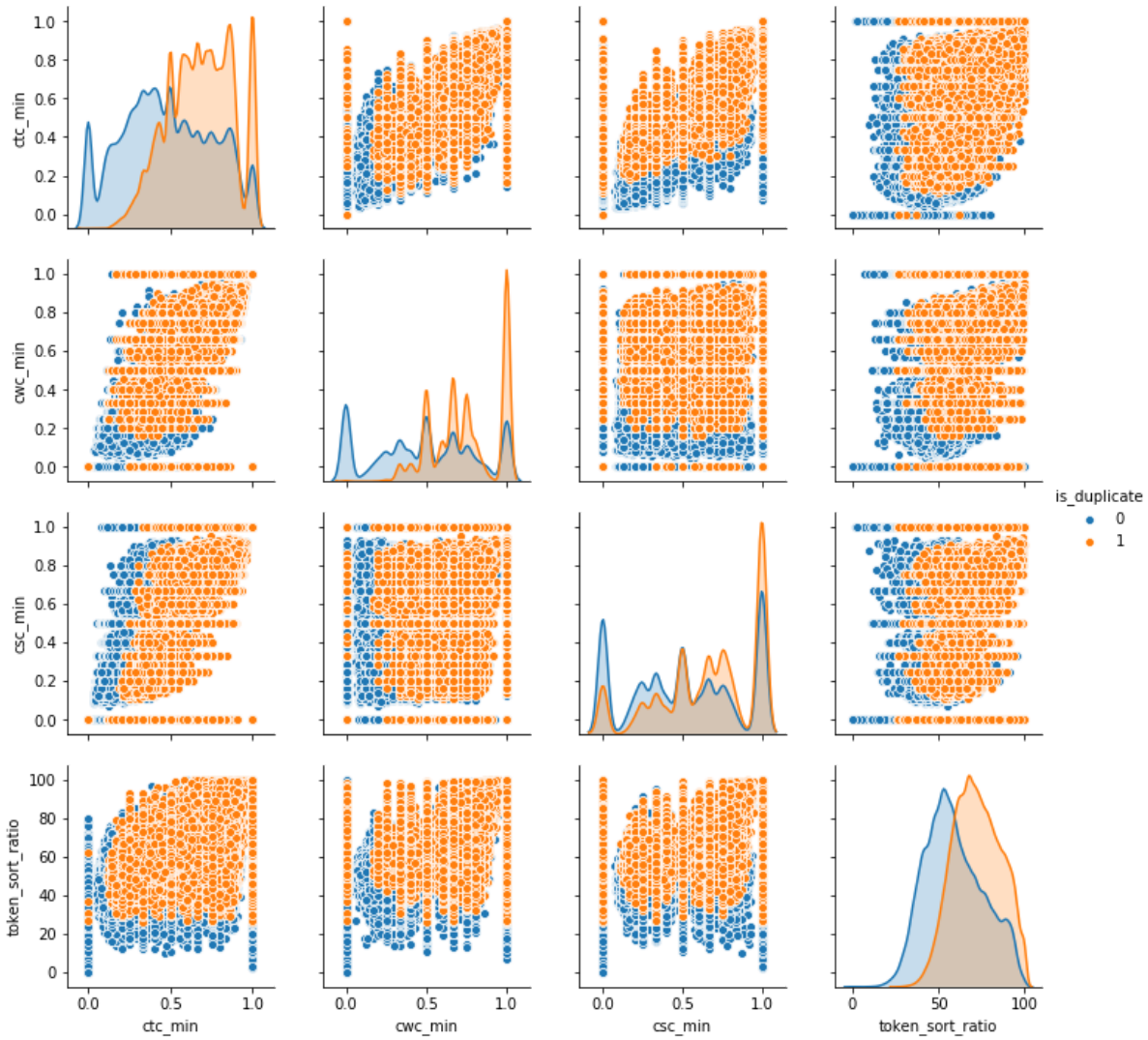
In [92]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=sto
pwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

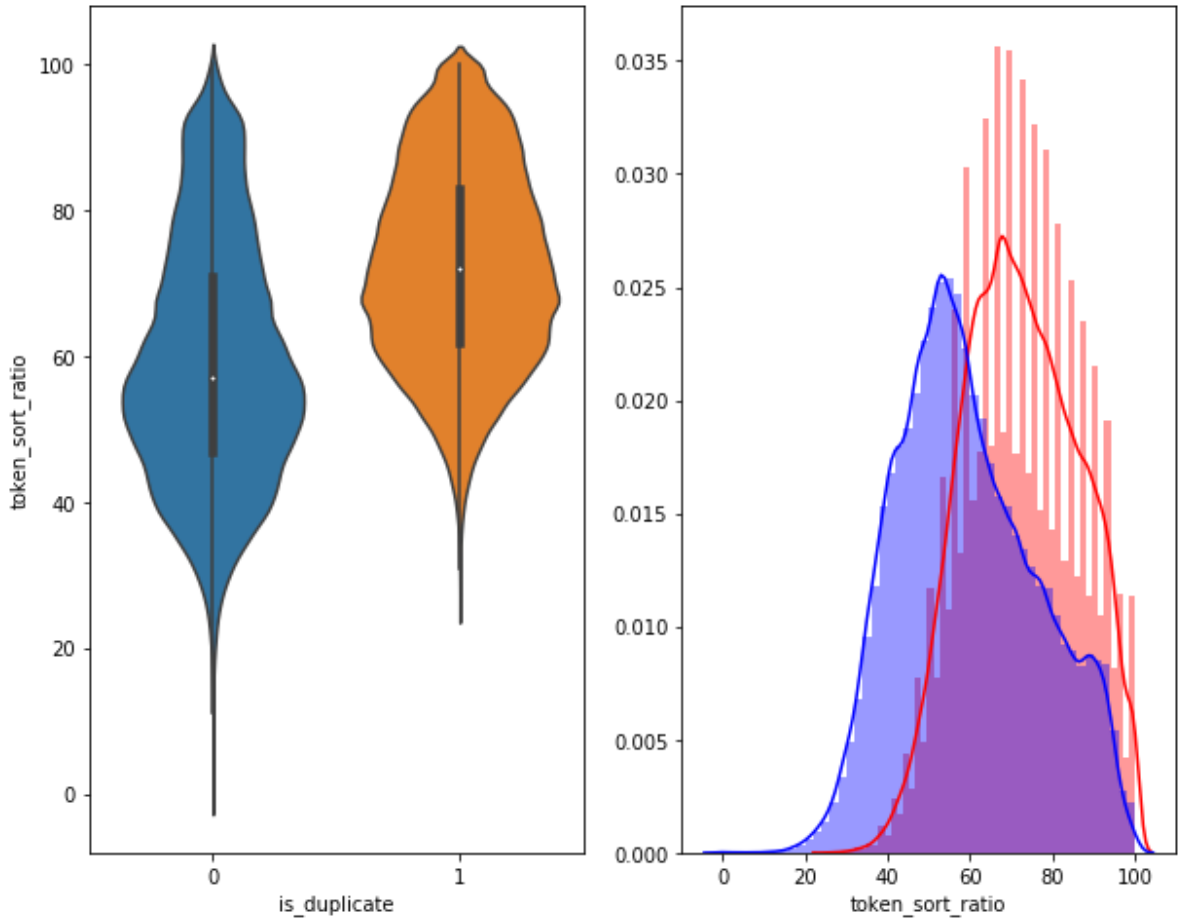
In [93]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stop
words)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word Cloud for non-Duplicate Question pairs:

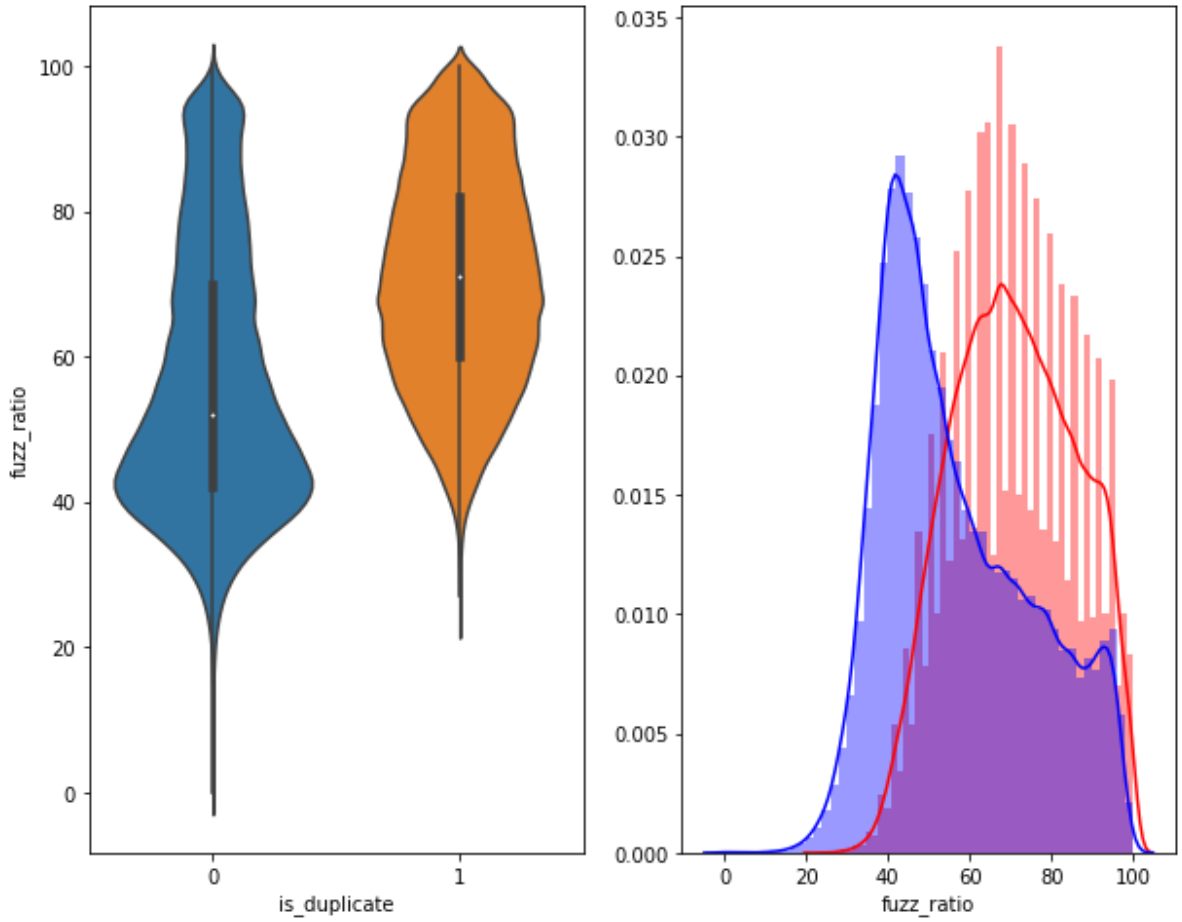
In [94]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```



In [95]:
```
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 [96]:
```
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", co
lor = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , c
olor = 'blue' )
plt.show()
```

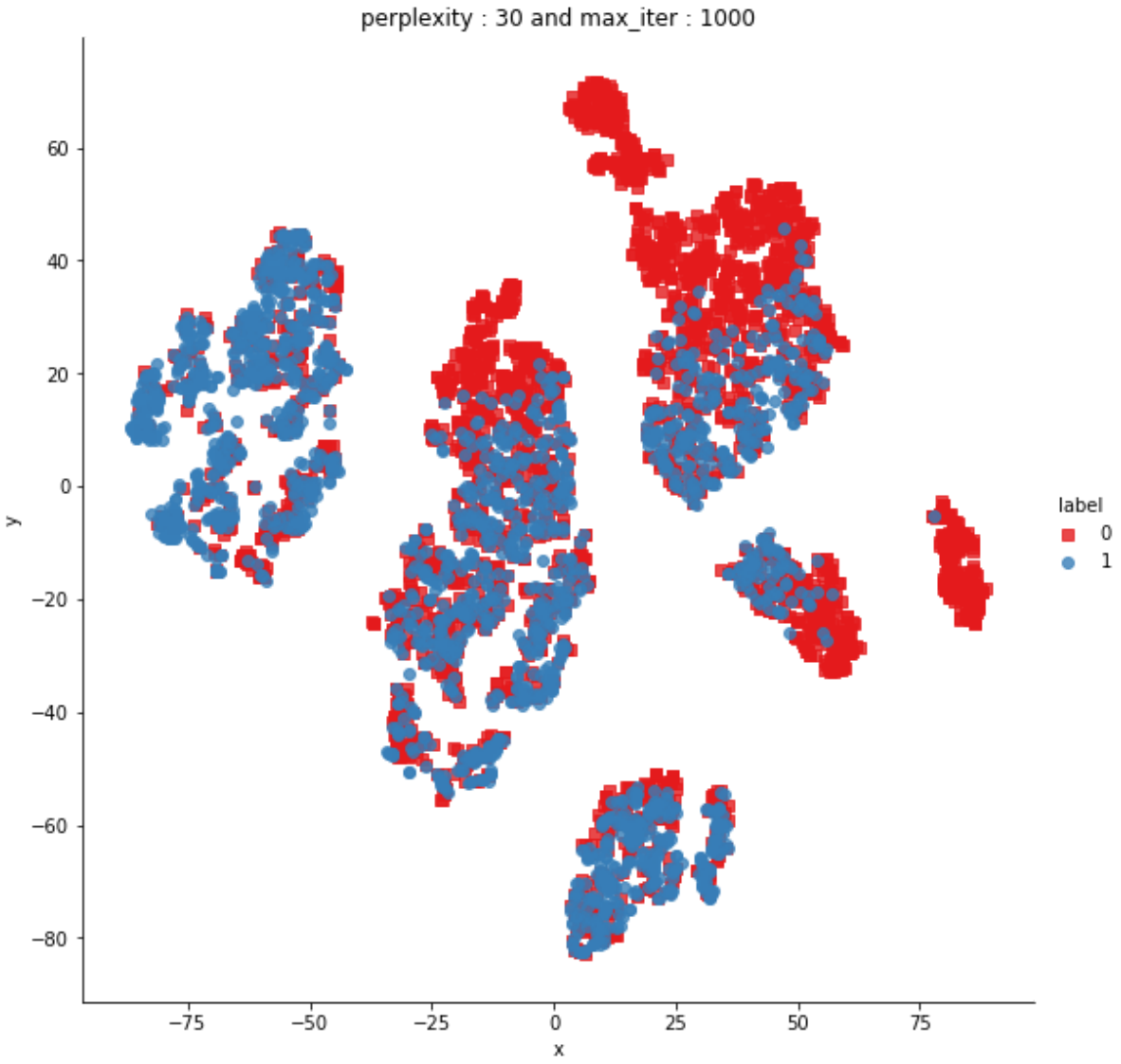


In [97]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after clea
ning the data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_mi
n', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_word_eq' , 'abs
_len_diff' , 'mean_len' , 'token_set_ratio' , 'token_sort_ratio' ,  'fuzz_rati
o' , 'fuzz_partial_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

In [98]:
```
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 0.250s...
[t-SNE] Computed neighbors for 5000 samples in 0.656s...
[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 0.422s
[t-SNE] Iteration 50: error = 80.8968964, gradient norm = 0.0430571 (50 itera
tions in 9.775s)
[t-SNE] Iteration 100: error = 70.3833160, gradient norm = 0.0099593 (50 iter
ations in 7.224s)
[t-SNE] Iteration 150: error = 68.6159134, gradient norm = 0.0056708 (50 iter
ations in 7.010s)
[t-SNE] Iteration 200: error = 67.7694321, gradient norm = 0.0040581 (50 iter
ations in 7.405s)
[t-SNE] Iteration 250: error = 67.2746048, gradient norm = 0.0033067 (50 iter
ations in 7.299s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.274605
[t-SNE] Iteration 300: error = 1.7729300, gradient norm = 0.0011900 (50 itera
tions in 7.821s)
[t-SNE] Iteration 350: error = 1.3714967, gradient norm = 0.0004818 (50 itera
tions in 7.624s)
[t-SNE] Iteration 400: error = 1.2036748, gradient norm = 0.0002779 (50 itera
tions in 7.722s)
[t-SNE] Iteration 450: error = 1.1132656, gradient norm = 0.0001889 (50 itera
tions in 7.511s)
[t-SNE] Iteration 500: error = 1.0582460, gradient norm = 0.0001434 (50 itera
tions in 7.740s)
[t-SNE] Iteration 550: error = 1.0222589, gradient norm = 0.0001180 (50 itera
tions in 7.639s)
[t-SNE] Iteration 600: error = 0.9984865, gradient norm = 0.0001015 (50 itera
tions in 7.713s)
[t-SNE] Iteration 650: error = 0.9830498, gradient norm = 0.0000958 (50 itera
tions in 7.552s)
[t-SNE] Iteration 700: error = 0.9726909, gradient norm = 0.0000877 (50 itera
tions in 7.740s)
[t-SNE] Iteration 750: error = 0.9647216, gradient norm = 0.0000823 (50 itera
tions in 7.727s)
[t-SNE] Iteration 800: error = 0.9582971, gradient norm = 0.0000755 (50 itera
tions in 8.264s)
[t-SNE] Iteration 850: error = 0.9531373, gradient norm = 0.0000697 (50 itera
tions in 8.276s)
[t-SNE] Iteration 900: error = 0.9484153, gradient norm = 0.0000696 (50 itera
tions in 8.911s)
[t-SNE] Iteration 950: error = 0.9445393, gradient norm = 0.0000659 (50 itera
tions in 8.213s)
[t-SNE] Iteration 1000: error = 0.9412127, gradient norm = 0.0000674 (50 iter
ations in 8.557s)
[t-SNE] Error after 1000 iterations: 0.941213
```

In [99]:
```
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,palett
e="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



In [100]:
```
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 0.031s...
[t-SNE] Computed neighbors for 5000 samples in 0.516s...
[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 0.344s
[t-SNE] Iteration 50: error = 80.3592682, gradient norm = 0.0335202 (50 itera
tions in 20.033s)
[t-SNE] Iteration 100: error = 69.1112671, gradient norm = 0.0036575 (50 iter
ations in 10.214s)
[t-SNE] Iteration 150: error = 67.6171112, gradient norm = 0.0017708 (50 iter
ations in 8.841s)
[t-SNE] Iteration 200: error = 67.0565109, gradient norm = 0.0011567 (50 iter
ations in 10.037s)
[t-SNE] Iteration 250: error = 66.7296524, gradient norm = 0.0009161 (50 iter
ations in 9.151s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729652
[t-SNE] Iteration 300: error = 1.4983541, gradient norm = 0.0006807 (50 itera
tions in 10.819s)
[t-SNE] Iteration 350: error = 1.1549147, gradient norm = 0.0001922 (50 itera
tions in 14.139s)
[t-SNE] Iteration 400: error = 1.0101781, gradient norm = 0.0000912 (50 itera
tions in 14.310s)
[t-SNE] Iteration 450: error = 0.9388669, gradient norm = 0.0000628 (50 itera
tions in 14.323s)
[t-SNE] Iteration 500: error = 0.9029322, gradient norm = 0.0000524 (50 itera
tions in 13.974s)
[t-SNE] Iteration 550: error = 0.8841860, gradient norm = 0.0000482 (50 itera
tions in 13.756s)
[t-SNE] Iteration 600: error = 0.8722453, gradient norm = 0.0000365 (50 itera
tions in 13.122s)
[t-SNE] Iteration 650: error = 0.8627461, gradient norm = 0.0000347 (50 itera
tions in 13.144s)
[t-SNE] Iteration 700: error = 0.8549610, gradient norm = 0.0000312 (50 itera
tions in 13.889s)
[t-SNE] Iteration 750: error = 0.8487639, gradient norm = 0.0000311 (50 itera
tions in 14.596s)
[t-SNE] Iteration 800: error = 0.8440317, gradient norm = 0.0000281 (50 itera
tions in 13.269s)
[t-SNE] Iteration 850: error = 0.8396705, gradient norm = 0.0000250 (50 itera
tions in 13.127s)
[t-SNE] Iteration 900: error = 0.8354425, gradient norm = 0.0000242 (50 itera
tions in 13.422s)
[t-SNE] Iteration 950: error = 0.8317489, gradient norm = 0.0000233 (50 itera
tions in 13.265s)
[t-SNE] Iteration 1000: error = 0.8288577, gradient norm = 0.0000257 (50 iter
ations in 13.398s)
[t-SNE] Error after 1000 iterations: 0.828858
```

In [101]:
```
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 feat
ures')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

In [15]:
```
# Load Basic Features
df_basic_feature = pd.read_csv("df_fe_without_preprocessing_train.csv",encodin
g='latin-1')
```

In [16]:
```
print("Columns : ",df_basic_feature.columns)
print("\nNumber of columns : ",len(df_basic_feature.columns))

df_basic_feature.head()
```

```
Columns :  Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplica
te',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_word
s',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q
2'],
      dtype='object')

Number of columns :  17
```

Out[16]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2le
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39

In [17]:
```
# Load Advance Features
df_advance_features = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
```

In [18]:
```
print("Columns : ",df_advance_features.columns)
print("\nNumber of columns : ",len(df_advance_features.columns))

df_advance_features.head()
```

```
Columns :  Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplica
te',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')

Number of columns :  21
```

Out[18]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_
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	0.99
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	0.59
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	0.24
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	0.00
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	0.66

5 rows × 21 columns

In [19]:
```
# Columns dropped from basic feature dataframe
df_basic_feature = df_basic_feature.drop(['qid1','qid2'],axis=1)

# Columns dropped from advance feature dataframe
df_advance_features = df_advance_features.drop(['qid1','qid2','question1','que
stion2','is_duplicate'],axis=1)

# Lets add both the truncated dataframe into one dataframe
df_basic_advance_features  = df_basic_feature.merge(df_advance_features, on='i
d',how='left')
```

In [20]:
```
nan_rows = df_basic_advance_features[df_basic_advance_features.isnull().any(1
)]
print (nan_rows)
```

```
           id                          question1  \
105780  105780    How can I develop android app?
201841  201841  How can I create an Android app?
363362  363362                              NaN

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

        freq_qid1  freq_qid2  q1len  q2len  q1_n_words  q2_n_words  \
105780          2          2     30      0           6           1
201841          1          2     32      0           7           1
363362          1          1      3    123           1          21

                ...          ctc_max  last_word_eq  first_word_eq  \
105780          ...              0.0           0.0            0.0
201841          ...              0.0           0.0            0.0
363362          ...              0.0           0.0            0.0

        abs_len_diff  mean_len  token_set_ratio  token_sort_ratio  fuzz_ratio
  \
105780           0.0       0.0                0                 0           0

201841           0.0       0.0                0                 0           0

363362          19.0      11.5                6                 5           5


        fuzz_partial_ratio  longest_substr_ratio
105780                   0                   0.0
201841                   0                   0.0
363362                  67                   0.5

[3 rows x 30 columns]
```

In [21]:
```
df_basic_advance_features = df_basic_advance_features[df_basic_advance_feature
s['question1'].notnull()]
df_basic_advance_features = df_basic_advance_features[df_basic_advance_feature
s['question2'].notnull()]
```

In [22]:
```
nan_rows = df_basic_advance_features[df_basic_advance_features.isnull().any(1
)]
print (nan_rows)
```

```
Empty DataFrame
Columns: [id, question1, question2, is_duplicate, freq_qid1, freq_qid2, q1le
n, q2len, q1_n_words, q2_n_words, word_Common, word_Total, word_share, freq_q
1+q2, freq_q1-q2, 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_r
atio, fuzz_ratio, fuzz_partial_ratio, longest_substr_ratio]
Index: []

[0 rows x 30 columns]
```

In [23]:
```
df_basic_advance_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404287 entries, 0 to 404289
Data columns (total 30 columns):
id                      404287 non-null int64
question1               404287 non-null object
question2               404287 non-null object
is_duplicate            404287 non-null int64
freq_qid1               404287 non-null int64
freq_qid2               404287 non-null int64
q1len                   404287 non-null int64
q2len                   404287 non-null int64
q1_n_words              404287 non-null int64
q2_n_words              404287 non-null int64
word_Common             404287 non-null float64
word_Total              404287 non-null float64
word_share              404287 non-null float64
freq_q1+q2              404287 non-null int64
freq_q1-q2              404287 non-null int64
cwc_min                 404287 non-null float64
cwc_max                 404287 non-null float64
csc_min                 404287 non-null float64
csc_max                 404287 non-null float64
ctc_min                 404287 non-null float64
ctc_max                 404287 non-null float64
last_word_eq            404287 non-null float64
first_word_eq           404287 non-null float64
abs_len_diff            404287 non-null float64
mean_len                404287 non-null float64
token_set_ratio         404287 non-null int64
token_sort_ratio        404287 non-null int64
fuzz_ratio              404287 non-null int64
fuzz_partial_ratio      404287 non-null int64
longest_substr_ratio    404287 non-null float64
dtypes: float64(14), int64(14), object(2)
memory usage: 95.6+ MB
```

In [24]:
```
print("Columns : ",df_basic_advance_features.columns)
print("\nNumber of columns : ",len(df_basic_advance_features.columns))

df_basic_advance_features.head()
```

```
Columns :  Index(['id', 'question1', 'question2', 'is_duplicate', 'freq_qid
1',
       'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')

Number of columns :  30
```

Out[24]:

	id	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_wo
0	0	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	14
1	1	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	8
2	2	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59	14
3	3	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
4	4	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	13

5 rows × 30 columns

In [25]:
```
target = df_basic_advance_features['is_duplicate']
```

In [26]:
```
df_basic_advance_features.drop(['id','is_duplicate'], axis=1, inplace=True)
```

In [27]:
```
print("Columns : ",df_basic_advance_features.columns)
print("\nNumber of columns : ",len(df_basic_advance_features.columns))

df_basic_advance_features.head()
```

```
Columns :  Index(['question1', 'question2', 'freq_qid1', 'freq_qid2', 'q1le
n', 'q2len',
       'q1_n_words', 'q2_n_words', 'word_Common', 'word_Total', 'word_share',
       'freq_q1+q2', 'freq_q1-q2', 'cwc_min', 'cwc_max', 'csc_min', 'csc_ma
x',
       'ctc_min', 'ctc_max', 'last_word_eq', 'first_word_eq', 'abs_len_diff',
       'mean_len', 'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')

Number of columns :  28
```

Out[27]:

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

5 rows × 28 columns

In [28]:
```
X_train,X_test, Y_train, Y_test = train_test_split(df_basic_advance_features,
target, stratify=target, test_size=0.3)
```

In [29]:
```
# Instanciate Tfidf Vectorizer
tfidf_q1 = TfidfVectorizer()

q1_train = tfidf_q1.fit_transform(X_train['question1'].values.astype('U'))
q1_test = tfidf_q1.transform(X_test['question1'].values.astype('U'))
```

In [30]:
```
print("Found {0} features from question1 column".format(len(tfidf_q1.get_featu
re_names())))
```

```
Found 58023 features from question1 column
```

In [31]:
```
tfidf_q2 = TfidfVectorizer()
q2_train = tfidf_q2.fit_transform(X_train['question2'].values.astype('U'))
q2_test = tfidf_q2.transform(X_test['question2'].values.astype('U'))
```

In [32]:
```
print("Found {0} features from question2 column".format(len(tfidf_q2.get_featu
re_names())))
```

```
Found 53616 features from question2 column
```

In [33]:
```
q1_q2_train = hstack((q1_train,q2_train))
q1_q2_test = hstack((q1_test,q2_test))
```

In [34]:
```
X_train.drop(['question1','question2'], axis=1, inplace=True)
X_test.drop(['question1','question2'], axis=1, inplace=True)
```

In [35]:
```
X_train = hstack((X_train, q1_q2_train),format="csr",dtype='float64')
X_test = hstack((X_test, q1_q2_test),format="csr",dtype='float64')
```

In [36]:
```
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 : (283000, 111665)
Number of data points in test data : (121287, 111665)
```

In [37]:
```
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:  0.6307985865724381 Class 1:  0.36920141342756185
---------- Distribution of output variable in train data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

In [38]:
```
print(X_train.shape)
print(X_test.shape)
```

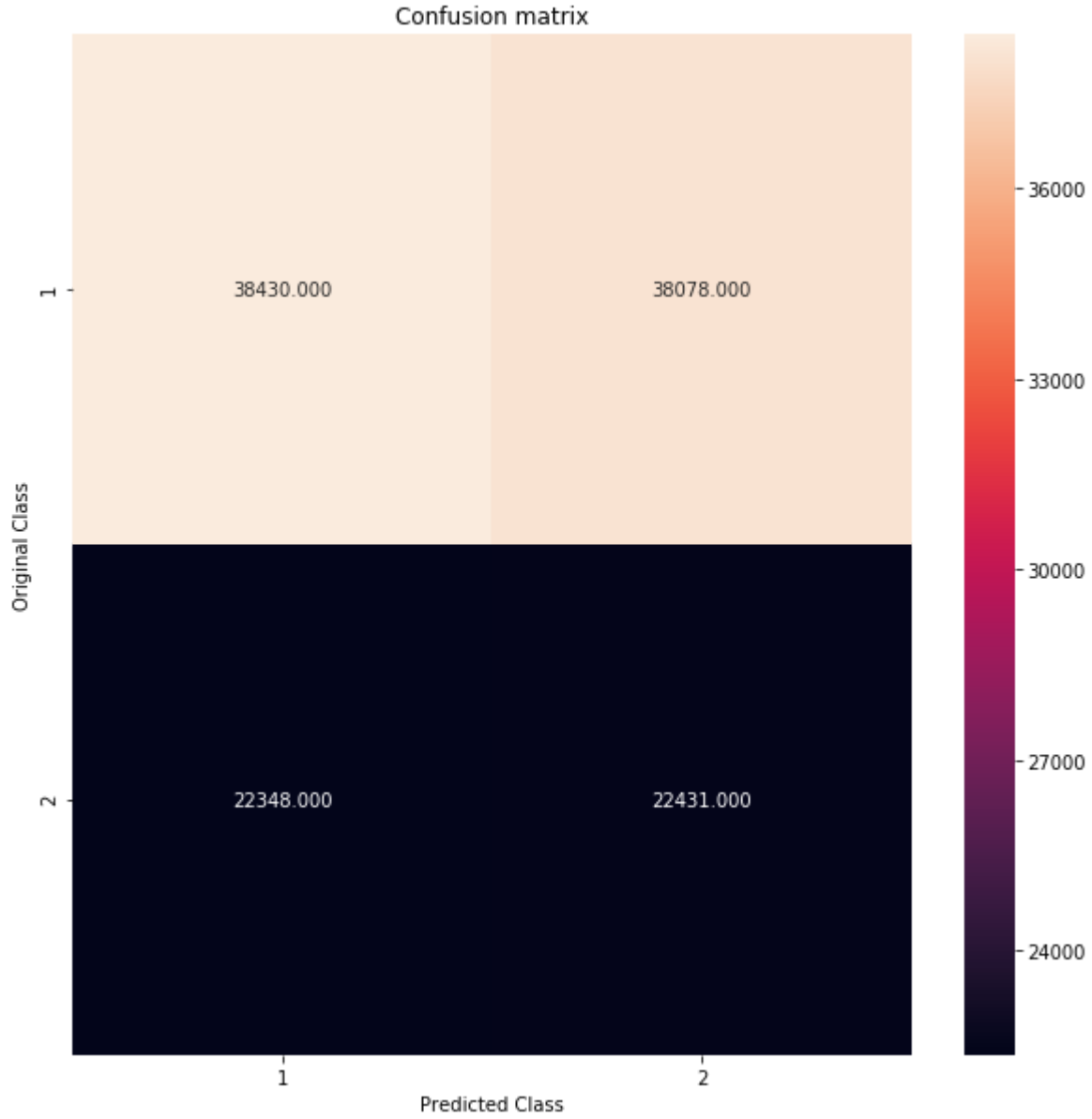
```
(283000, 111665)
(121287, 111665)
```

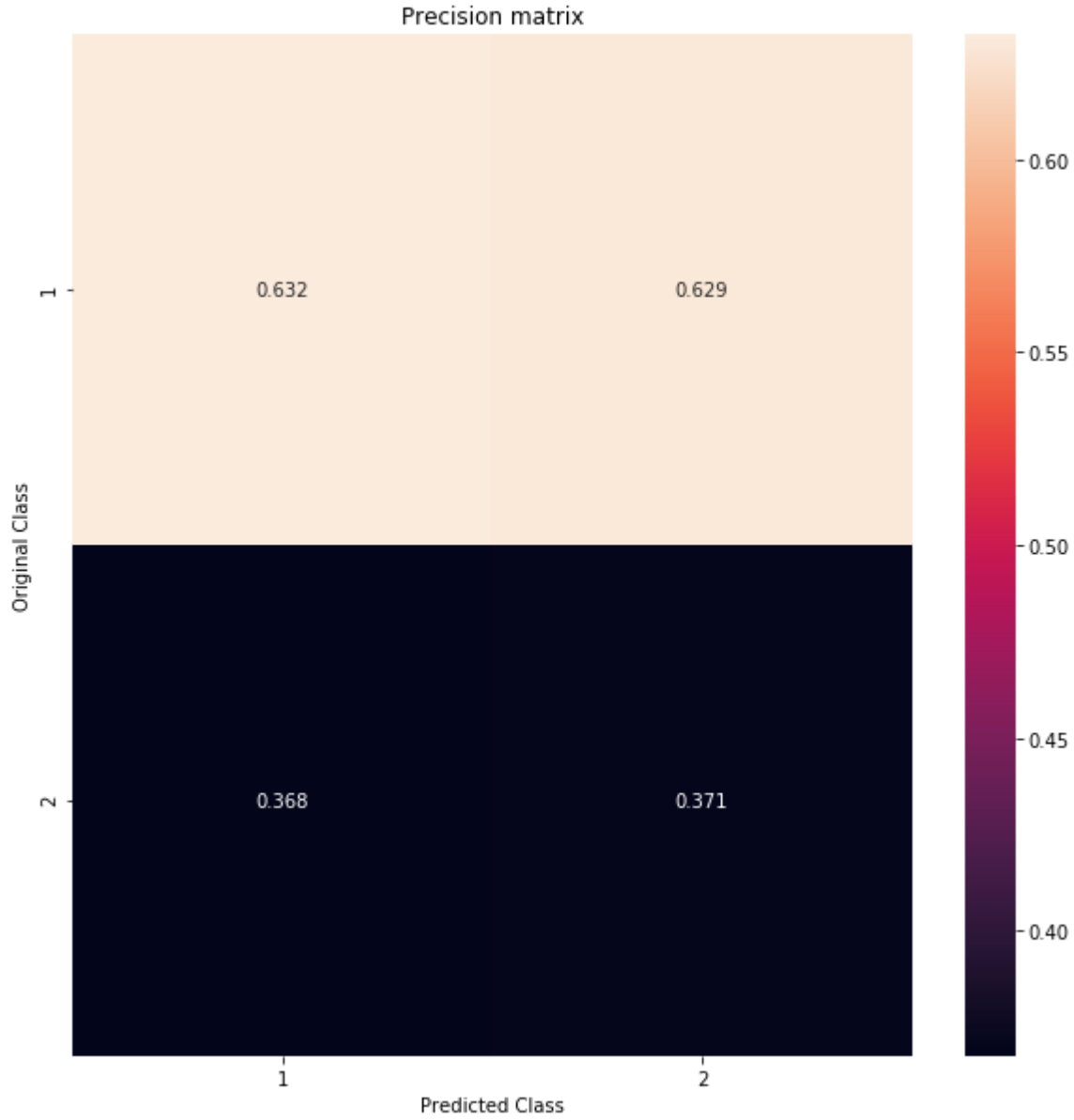
In [39]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    plt.figure(figsize=(10,10))
    sns.heatmap(C, annot=True, fmt=".3f", xticklabels=labels, yticklabels=labe
ls)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")
    plt.show()
    plt.figure(figsize=(10,10))
    sns.heatmap(B, annot=True,fmt=".3f", xticklabels=labels, yticklabels=label
s)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")
    plt.show()
    plt.figure(figsize=(10,10))
    # representing B in heatmap format
    sns.heatmap(A, annot=True, fmt=".3f", xticklabels=labels, yticklabels=labe
ls)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

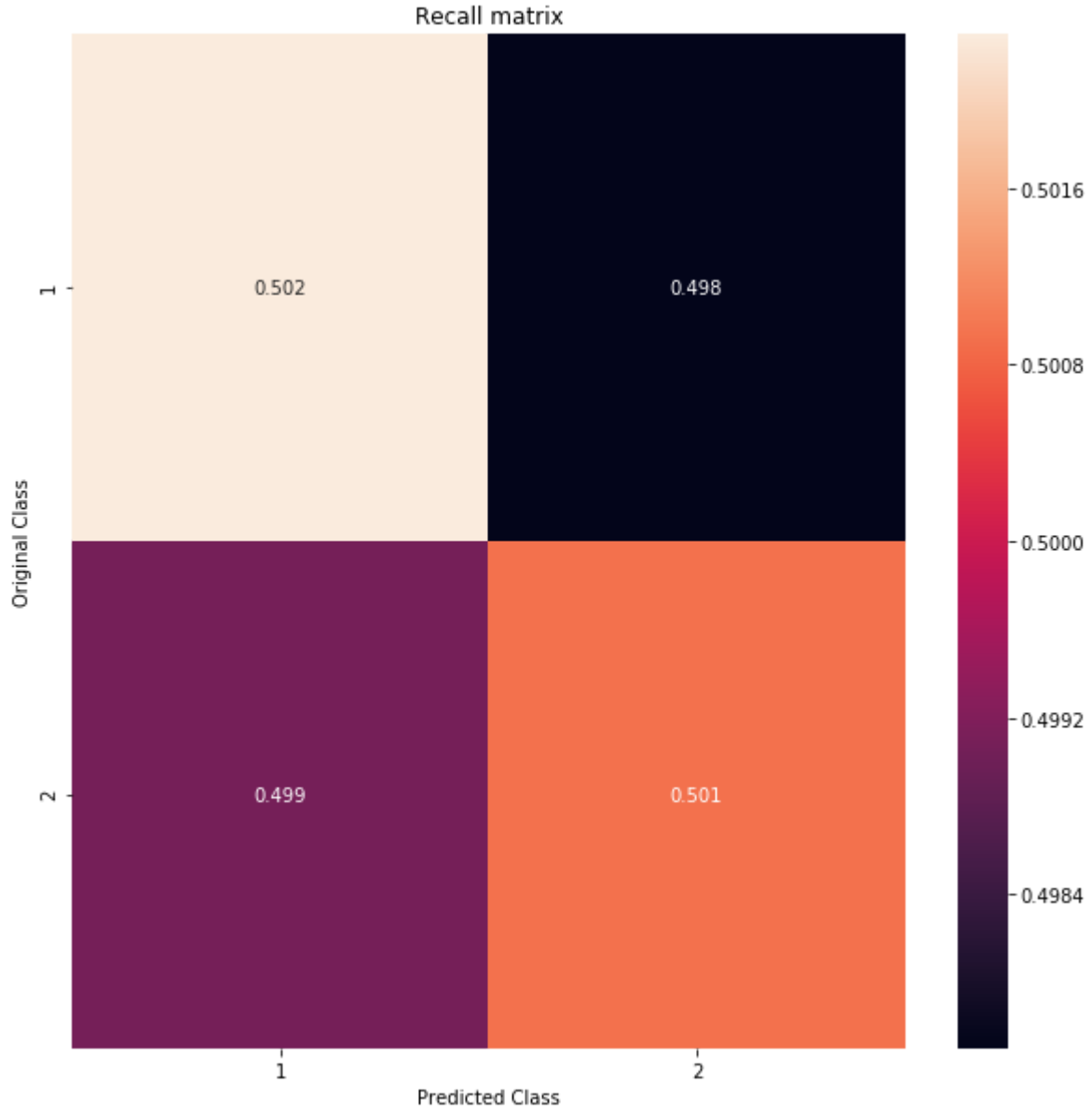
Random Model

In [127]:
```
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 0.8833916281083102

<Figure size 1440x288 with 0 Axes>
```



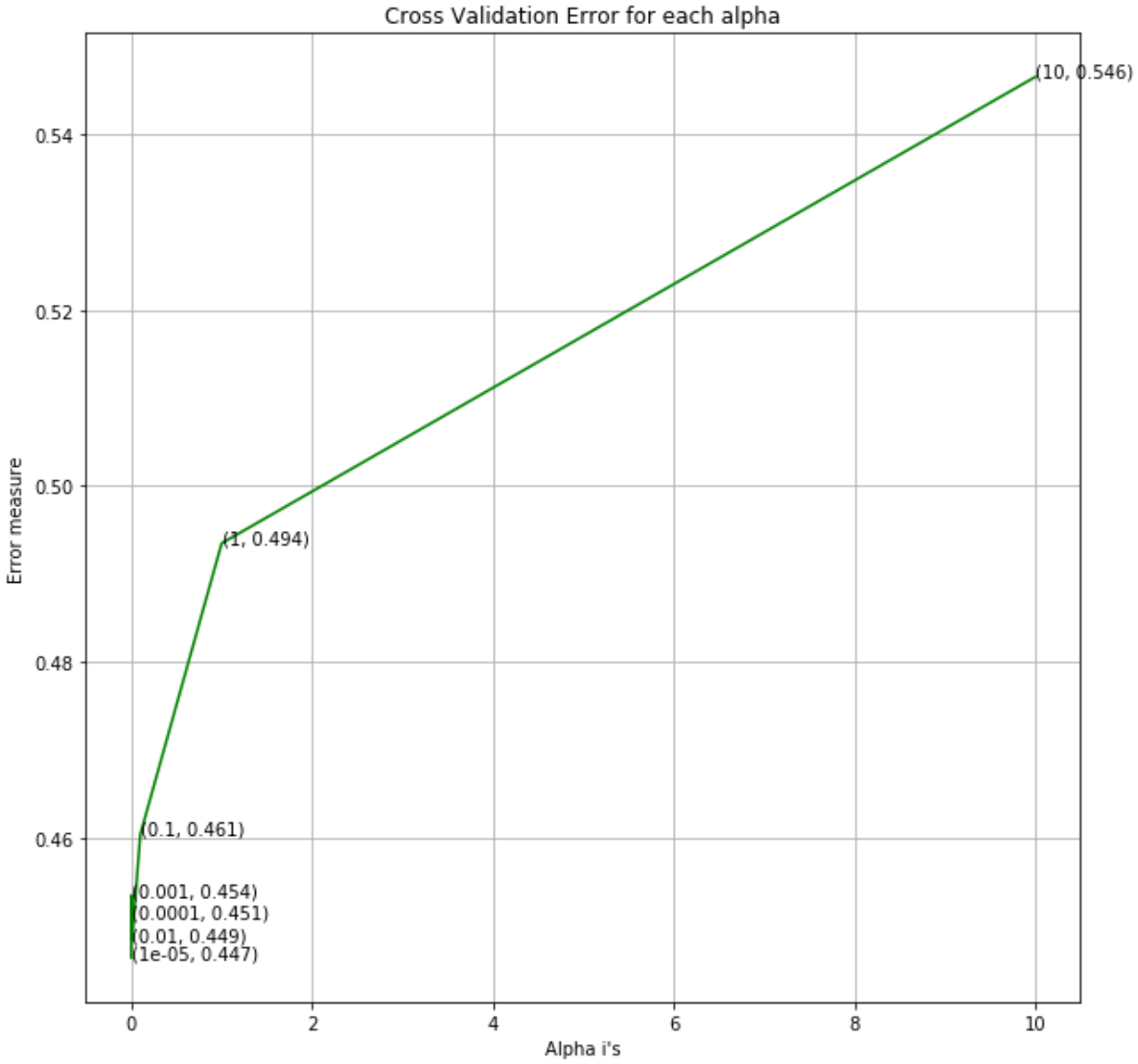

Precision matrix
0.632
0.629
0.368
0.371
1
2
Original Class
1
2
Predicted Class
0.60
0.55
0.50
0.45
0.40



Logistic Regression with Hyperparameter Tuning

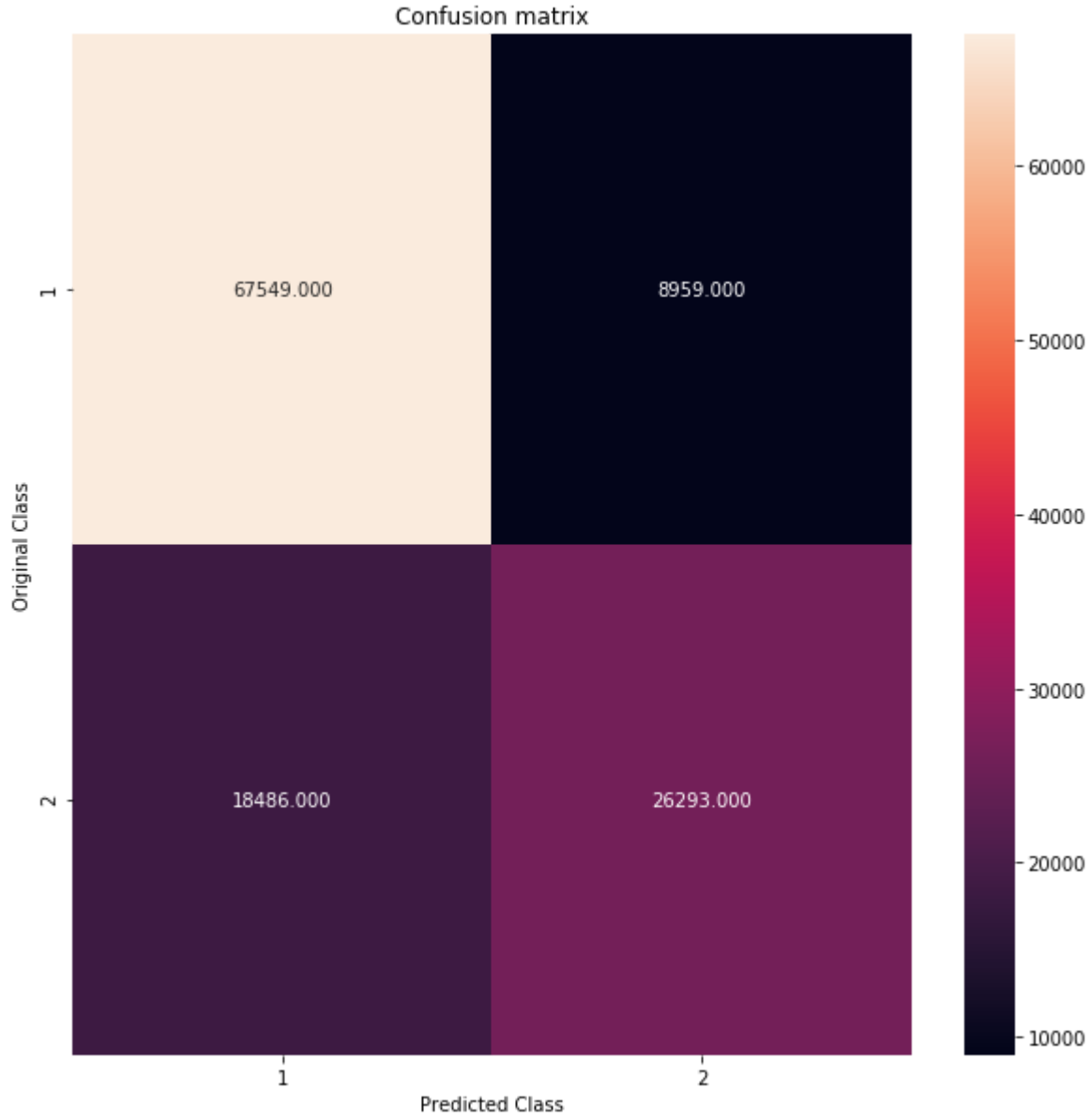
In [128]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=0)
    clf.fit(X_train, Y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, 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(figsize=(10,10))
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', rand
om_state=0)
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)
```

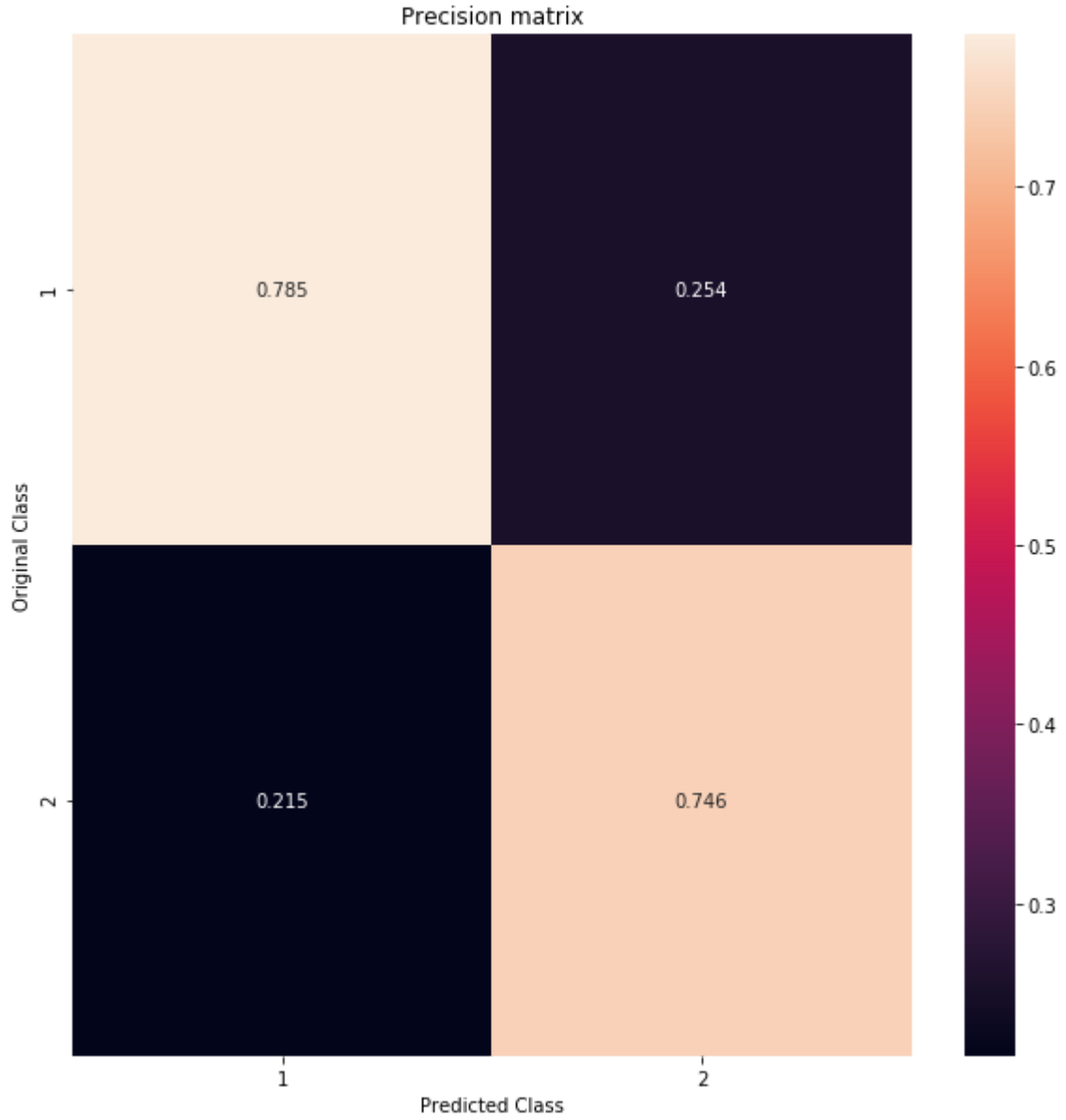
```
For values of alpha =  1e-05 The log loss is: 0.4465351908582037
For values of alpha =  0.0001 The log loss is: 0.4510708038714438
For values of alpha =  0.001 The log loss is: 0.4536125543143072
For values of alpha =  0.01 The log loss is: 0.44855620310412886
For values of alpha =  0.1 The log loss is: 0.46050948678278414
For values of alpha =  1 The log loss is: 0.4935122470809026
For values of alpha =  10 The log loss is: 0.5463893581149573
```

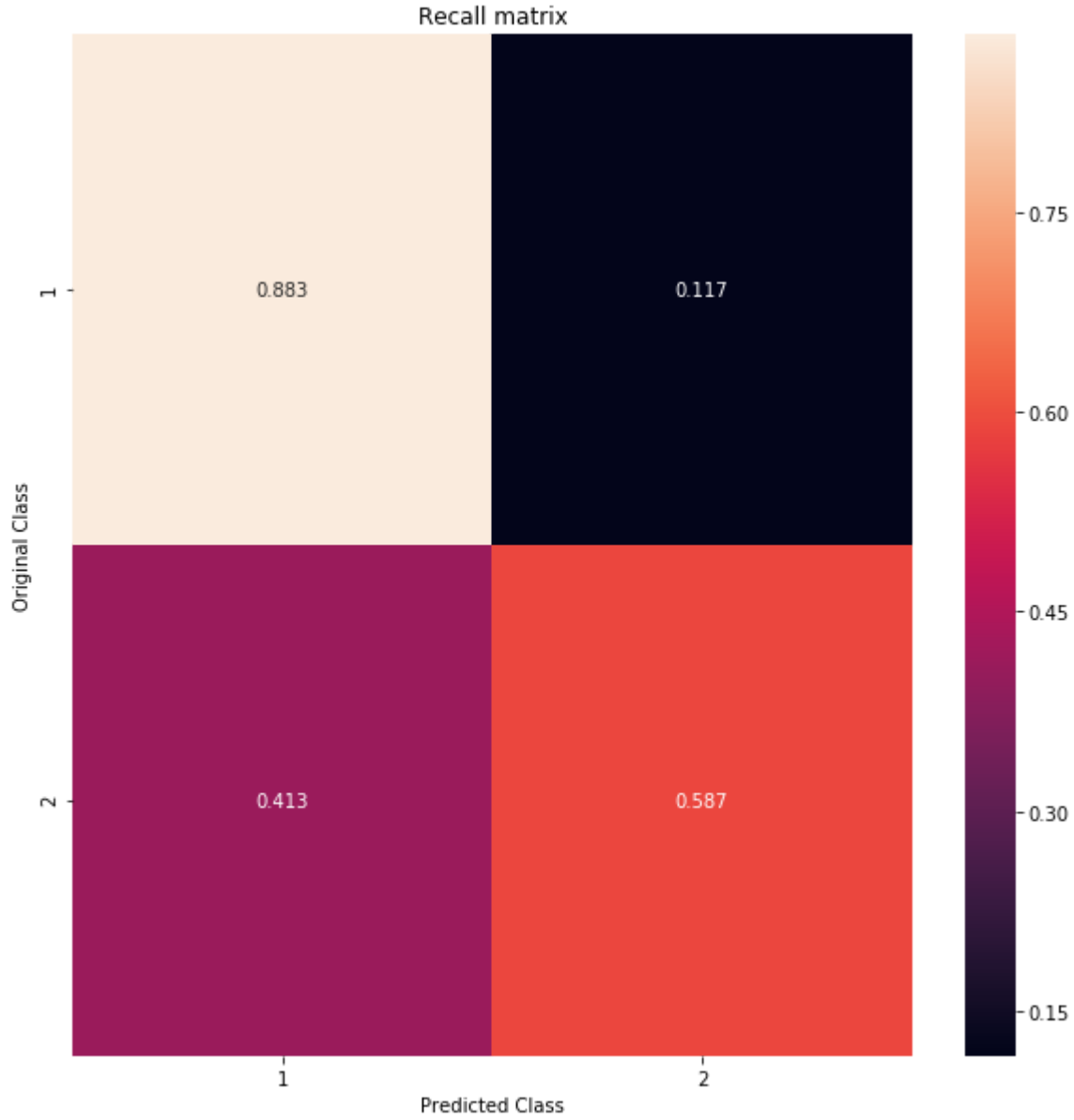



```
For values of best alpha =  1e-05 The train log loss is: 0.44405970518027327
For values of best alpha =  1e-05 The test log loss is: 0.4465351908582037
Total number of data points : 121287
```

```
<Figure size 1440x288 with 0 Axes>
```


Confusion matrix
67549.000
8959.000
18486.000
26293.000
Original Class
Predicted Class
1
2
60000
50000
40000
30000
20000
10000

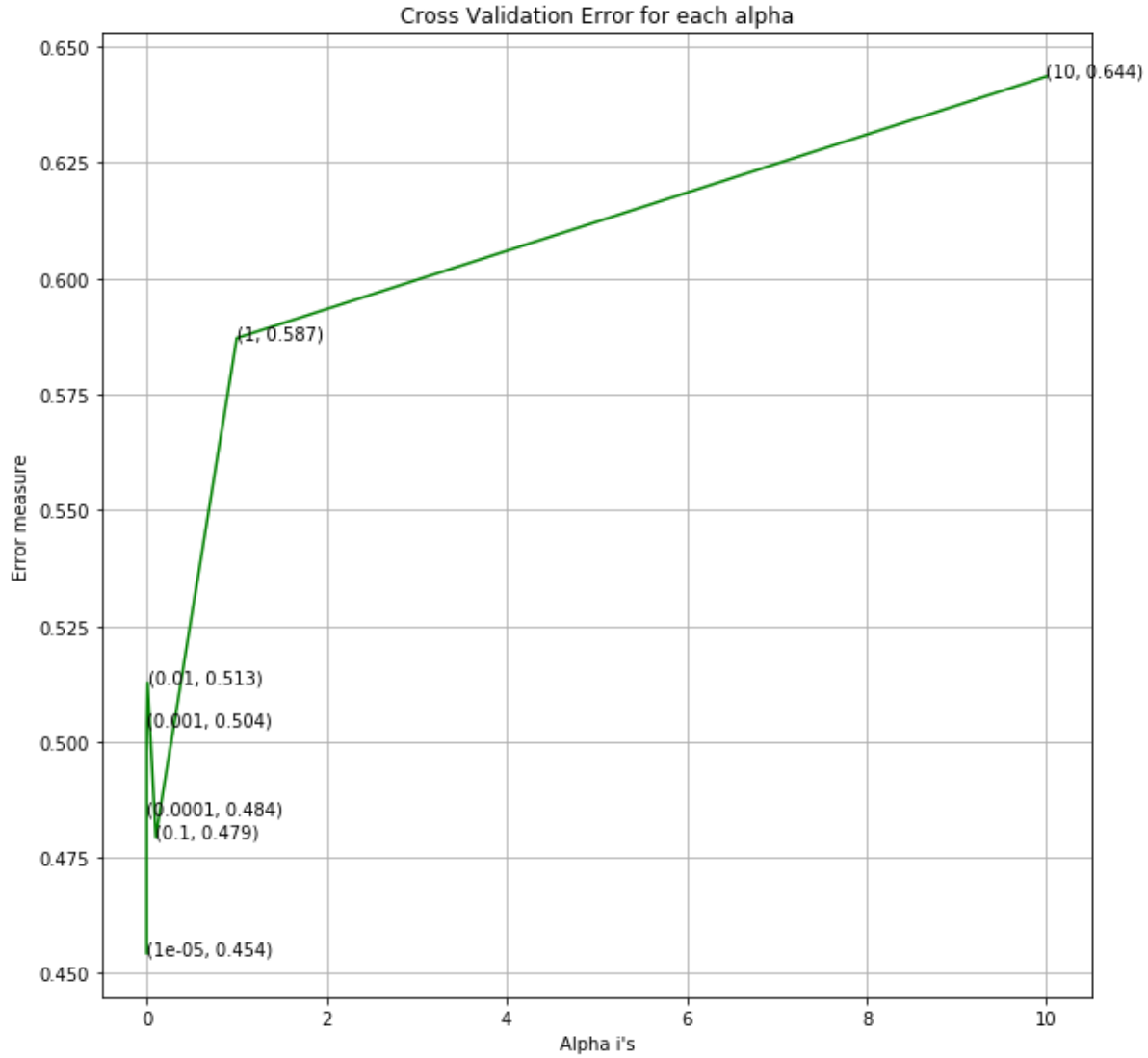

Precision matrix
0.785
0.254
0.215
0.746
Original Class
Predicted Class
1
2
0.7
0.6
0.5
0.4
0.3



Linear SVM with Hyperparameter Tuning

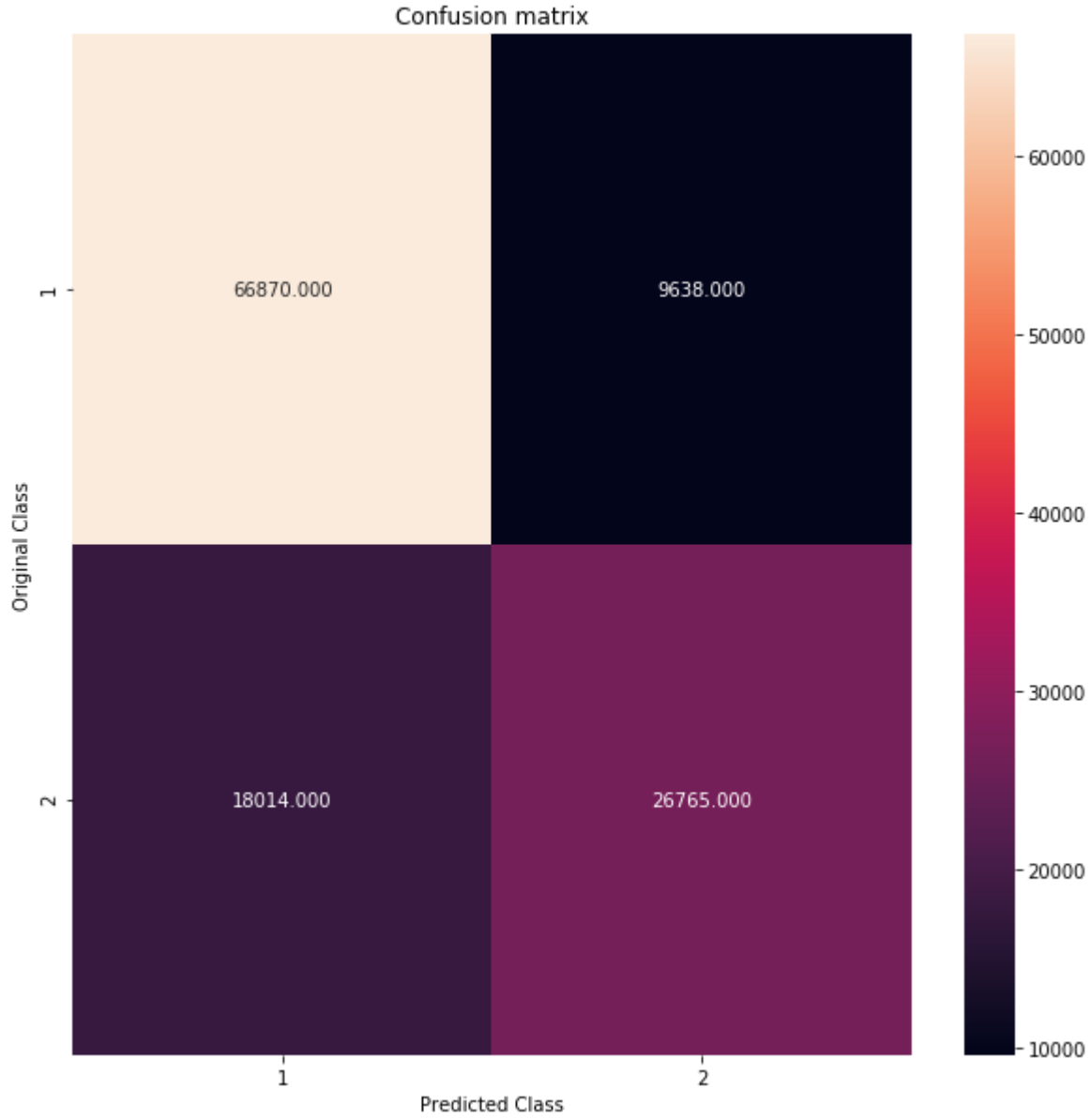
In [129]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=0
)
    clf.fit(X_train, Y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, 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(figsize=(10,10))
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', ra
ndom_state=0)
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)
```

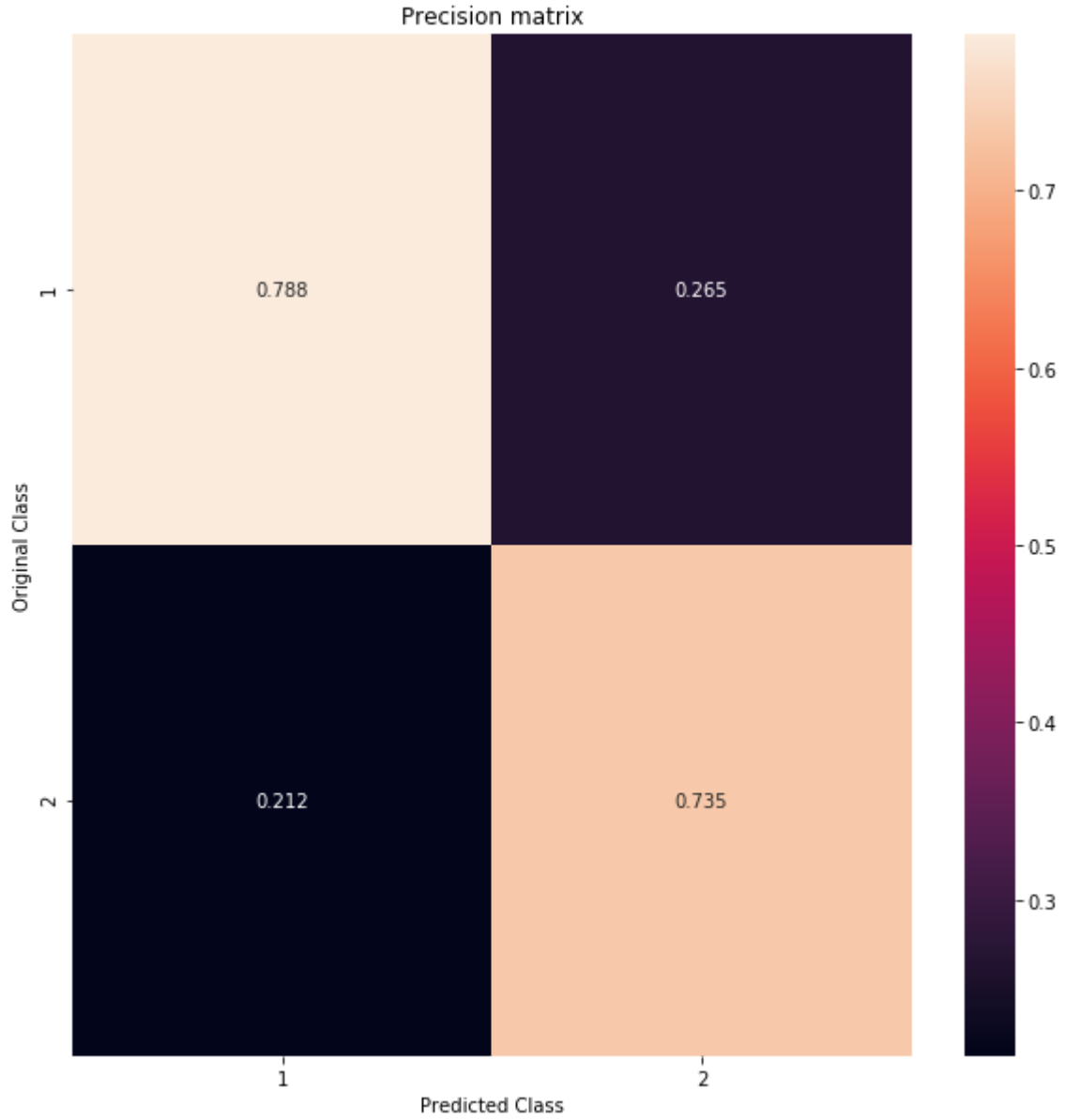
```
For values of alpha =  1e-05 The log loss is: 0.4543311739540758
For values of alpha =  0.0001 The log loss is: 0.4844035484643641
For values of alpha =  0.001 The log loss is: 0.5035892506898417
For values of alpha =  0.01 The log loss is: 0.5127613205614597
For values of alpha =  0.1 The log loss is: 0.47938839063137406
For values of alpha =  1 The log loss is: 0.587114114674345
For values of alpha =  10 The log loss is: 0.6435968666495386
```

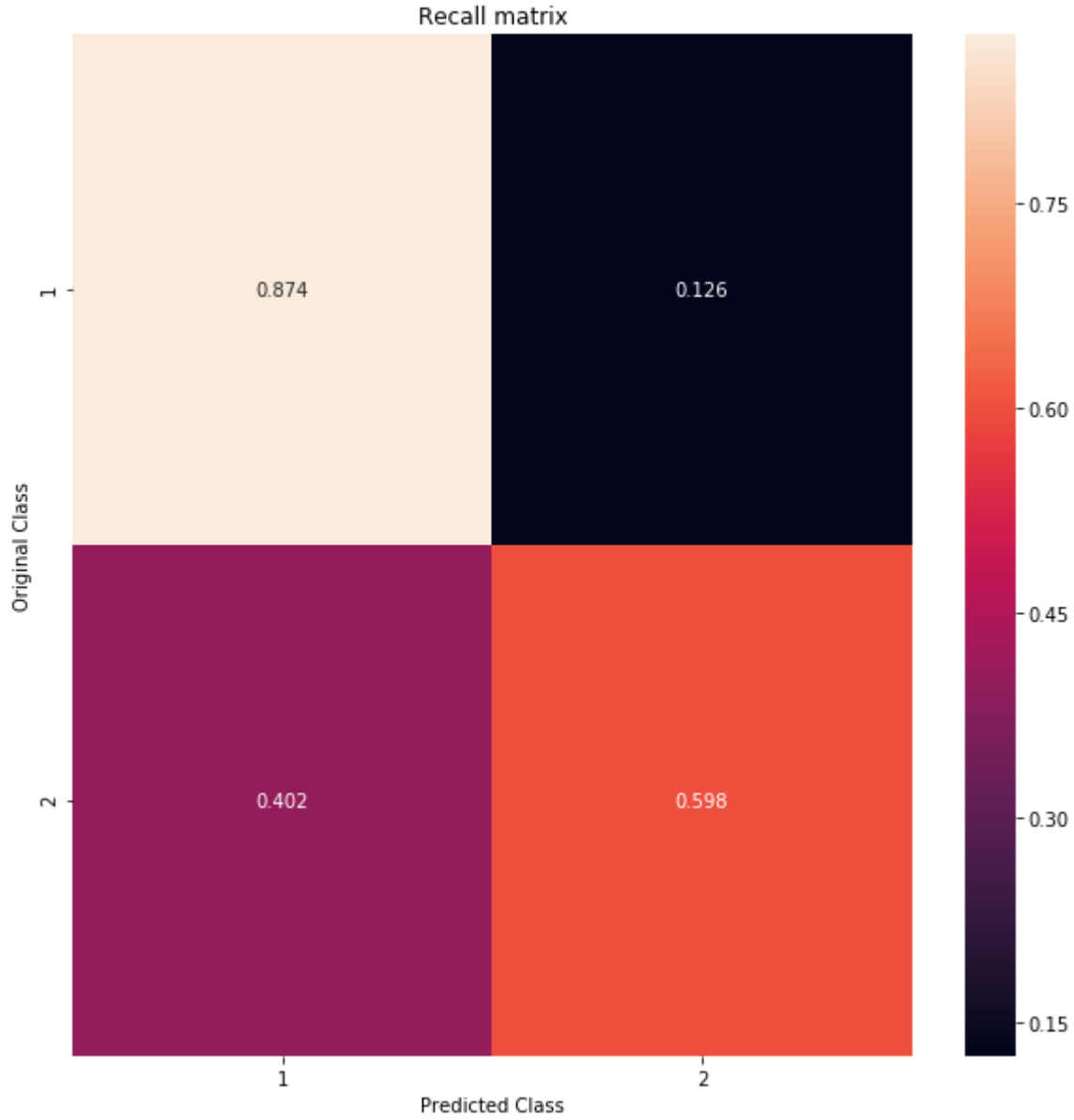



```
For values of best alpha =  1e-05 The train log loss is: 0.45178380451545314
For values of best alpha =  1e-05 The test log loss is: 0.4543311739540758
Total number of data points : 121287
```

```
<Figure size 1440x288 with 0 Axes>
```


Confusion matrix
66870.000
9638.000
18014.000
26765.000
Original Class
Predicted Class
1
2
60000
50000
40000
30000
20000
10000


Precision matrix
0.788
0.265
0.212
0.735
Original Class
Predicted Class
1
2
0.7
0.6
0.5
0.4
0.3


Recall matrix
0.874
0.126
0.402
0.598
1
2
Original Class
1
2
Predicted Class
0.75
0.60
0.45
0.30
0.15

In [40]:
```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV

def hyperparameter_tunning(X,Y):
    params = {'n_estimators' : [1,2,4,6,8,10,15,20,30,40,60,80,100,125,150,
175,200,250,300],'learning_rate' :[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]}
    param_grid = params
    model = XGBClassifier(nthread=-1)
    kfold = StratifiedKFold(n_splits=5, shuffle=True)
    random_search = RandomizedSearchCV(model, param_grid, scoring="neg_log_
loss", n_jobs=-1, cv=kfold)
    random_result = random_search.fit(X,Y)
    # Summarize results
    print("Best: %f using %s" % (random_result.best_score_, random_result.b
est_params_))
    print()
    means = random_result.cv_results_['mean_test_score']
    stds = random_result.cv_results_['std_test_score']
    params = random_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))
    return random_result
```

In [41]:
```
start = dt.datetime.now()
# Tune hyperparameter values
random_result = hyperparameter_tunning(X_train, Y_train)
print("\nTimeTaken: ",dt.datetime.now() - start)
```

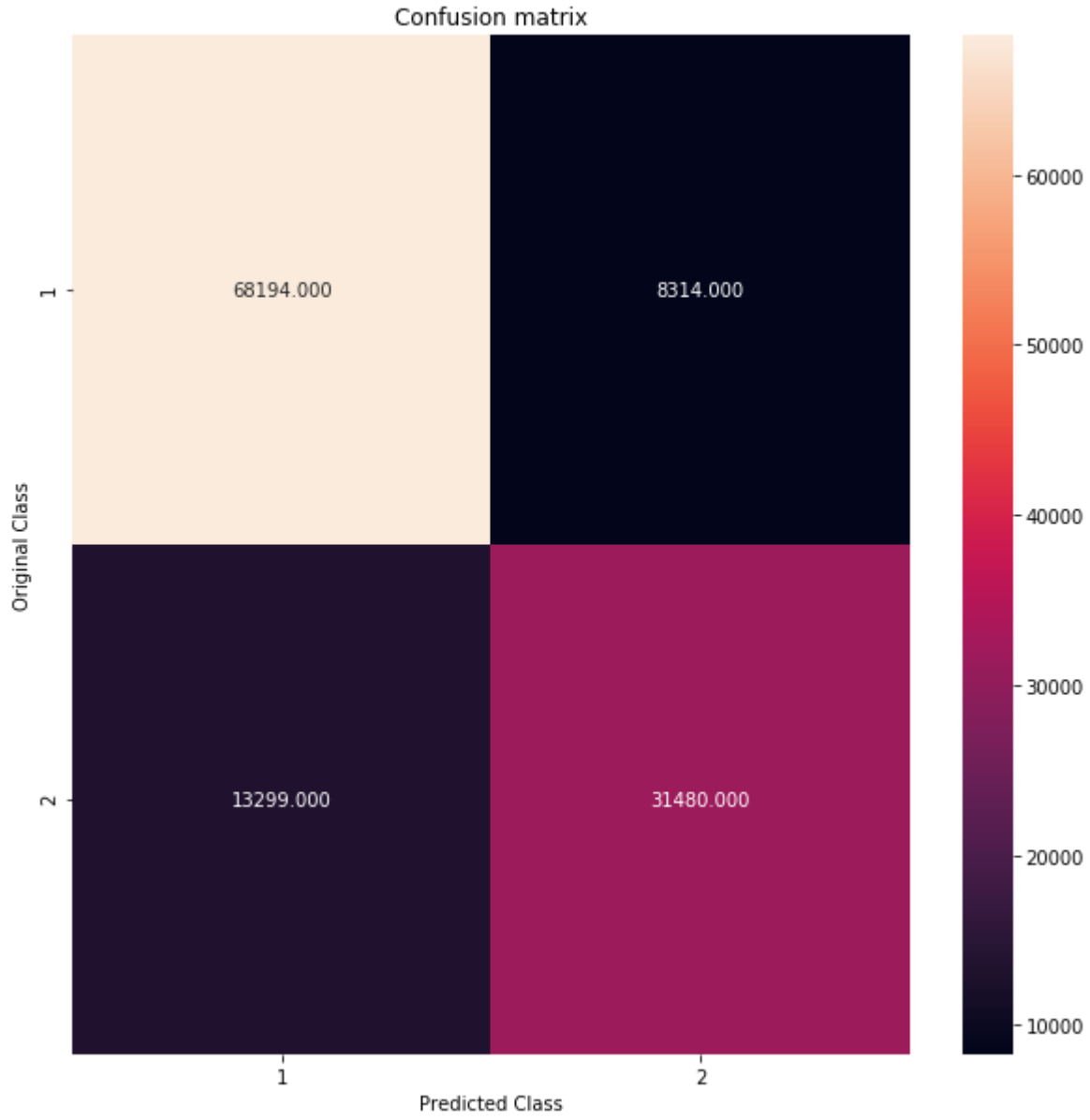
```
Best: -0.356765 using {'n_estimators': 60, 'learning_rate': 0.2}

-0.434575 (0.001258) with: {'n_estimators': 6, 'learning_rate': 0.3}
-0.692392 (0.000003) with: {'n_estimators': 2, 'learning_rate': 0.001}
-0.356765 (0.001227) with: {'n_estimators': 60, 'learning_rate': 0.2}
-0.427077 (0.001445) with: {'n_estimators': 10, 'learning_rate': 0.2}
-0.402497 (0.001583) with: {'n_estimators': 30, 'learning_rate': 0.1}
-0.539496 (0.000453) with: {'n_estimators': 6, 'learning_rate': 0.1}
-0.469423 (0.001232) with: {'n_estimators': 6, 'learning_rate': 0.2}
-0.682080 (0.000080) with: {'n_estimators': 30, 'learning_rate': 0.001}
-0.657830 (0.000200) with: {'n_estimators': 100, 'learning_rate': 0.001}
-0.367143 (0.002219) with: {'n_estimators': 80, 'learning_rate': 0.1}

TimeTaken:  0:28:45.221896
```

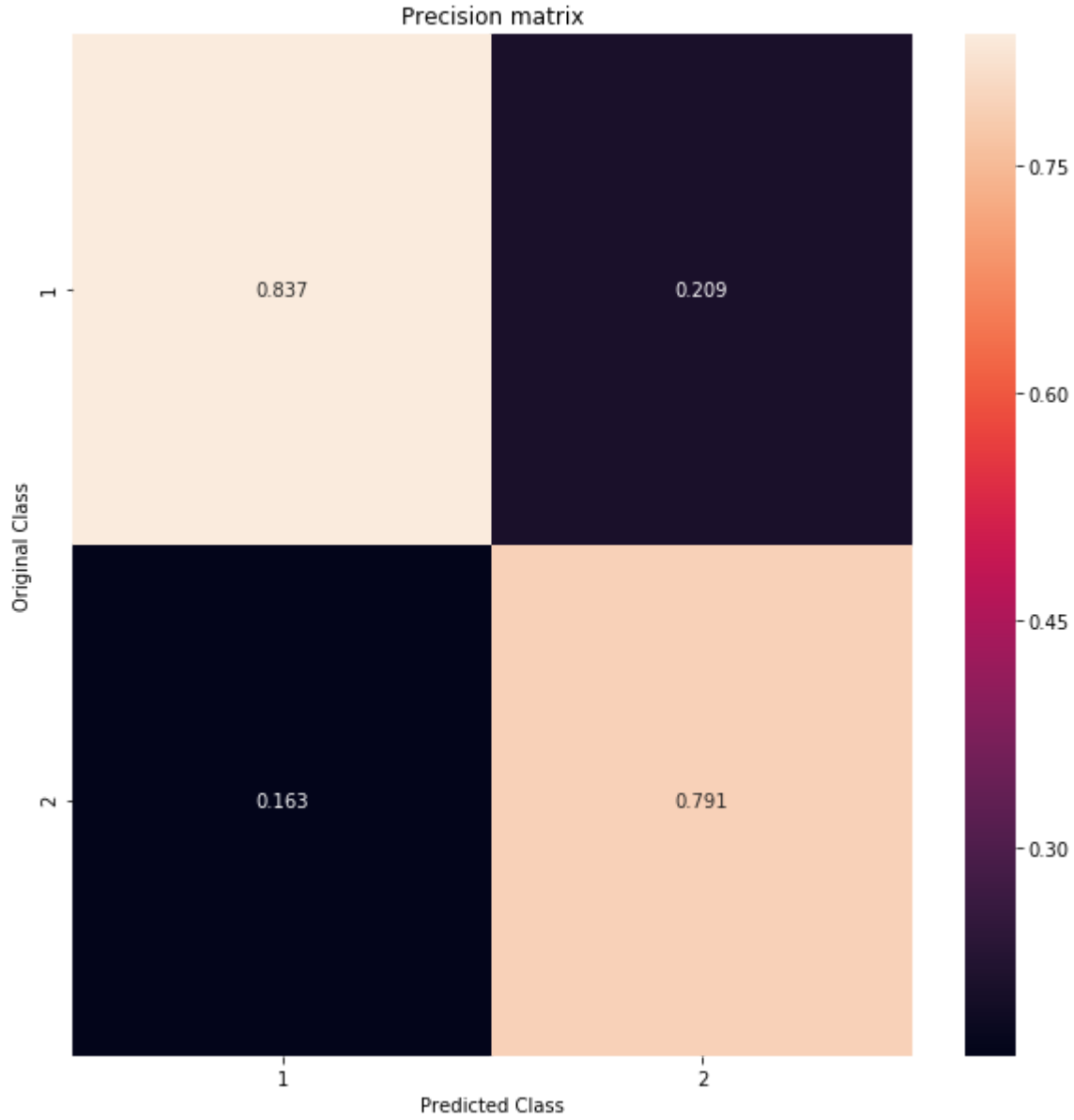
In [46]:
```
xGBClassifier = XGBClassifier(learning_rate=0.2, n_estimators=60,n_jobs=-1)
```

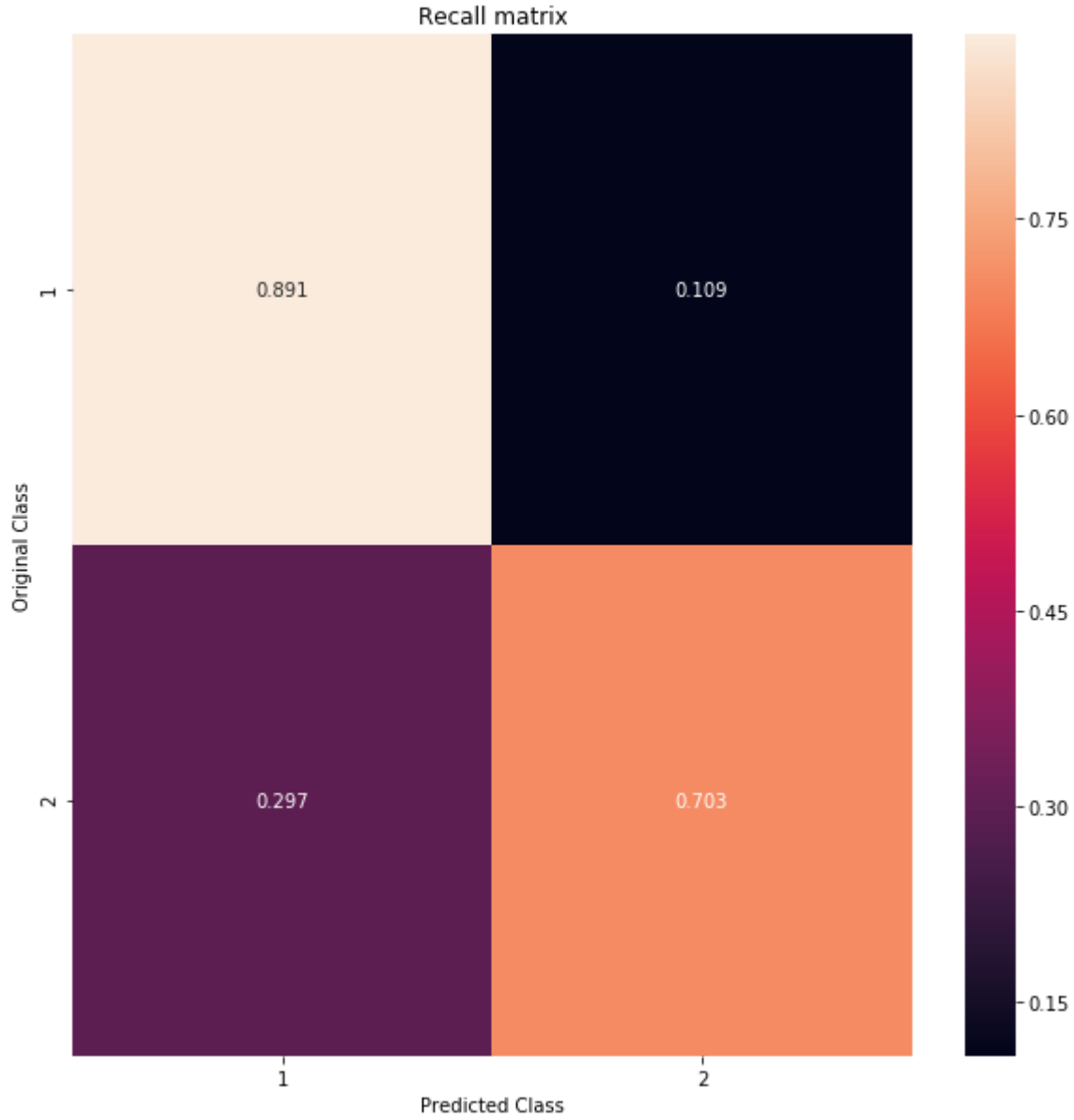
In [47]:
```
xGBClassifier.fit(X_train, Y_train)
predict_y = xGBClassifier.predict_proba(X_train)
print("The train log loss is:",log_loss(Y_train, predict_y, labels=xGBClass
ifier.classes_, eps=1e-15))
predict_y = xGBClassifier.predict_proba(X_test)
print("The test log loss is:",log_loss(Y_test, predict_y, labels=xGBClassif
ier.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)
```

```
The train log loss is: 0.356488830425697
The test log loss is: 0.3600268694576062
Total number of data points : 121287
```

```
<Figure size 1440x288 with 0 Axes>
```




Precision matrix
0.837
0.209
0.163
0.791
Original Class
Predicted Class
1
2
0.75
0.60
0.45
0.30



In [48]:
```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ['Model Name','Train log loss ','Test Log Loss']
ptable.add_row(["Random","NA","0.88"])
ptable.add_row(["Logistic Regression","0.44","0.45"])
ptable.add_row(["Linear SVM","0.44","0.44"])
ptable.add_row(["xgboost","0.35","0.36"])
print(ptable)
```

```
+---------------------+-----------------+---------------+
|      Model Name     | Train log loss  | Test Log Loss |
+---------------------+-----------------+---------------+
|        Random       |        NA       |      0.88     |
| Logistic Regression |       0.44      |      0.45     |
|      Linear SVM     |       0.44      |      0.44     |
|       xgboost       |       0.35      |      0.36     |
+---------------------+-----------------+---------------+
```