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.

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 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 s
tep 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 Ind
ian 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 comm
ents?","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

Metric(s):

log-loss
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 [2]:
```
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
```

3.1 Reading data and basic stats

In [3]:
```
df = pd.read_csv("train.csv")

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

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

In [4]:
```
df.head()
```

Out[4]:

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

In [5]:
```
df.info()
```

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

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

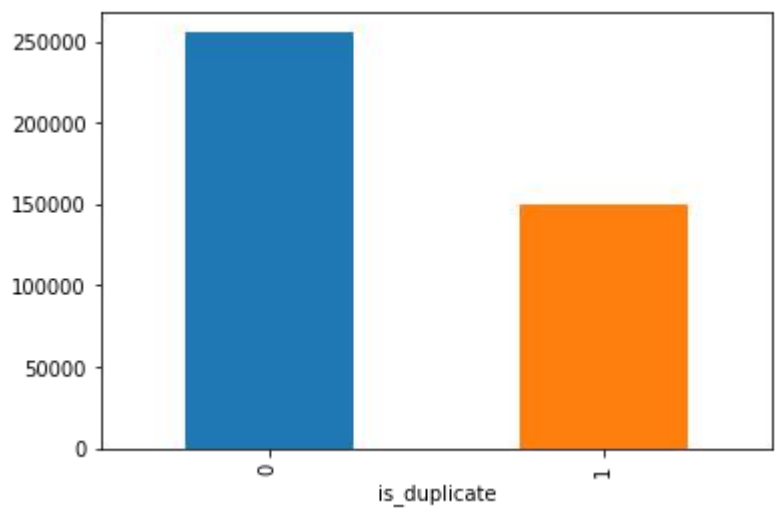
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2 Distribution of data points among output classes

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

In [6]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1c90cf3d9e8>



In [7]:
```
print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

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

In [8]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round(df['is_duplic
ate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_duplicate'].me
an()*100, 2)))
```

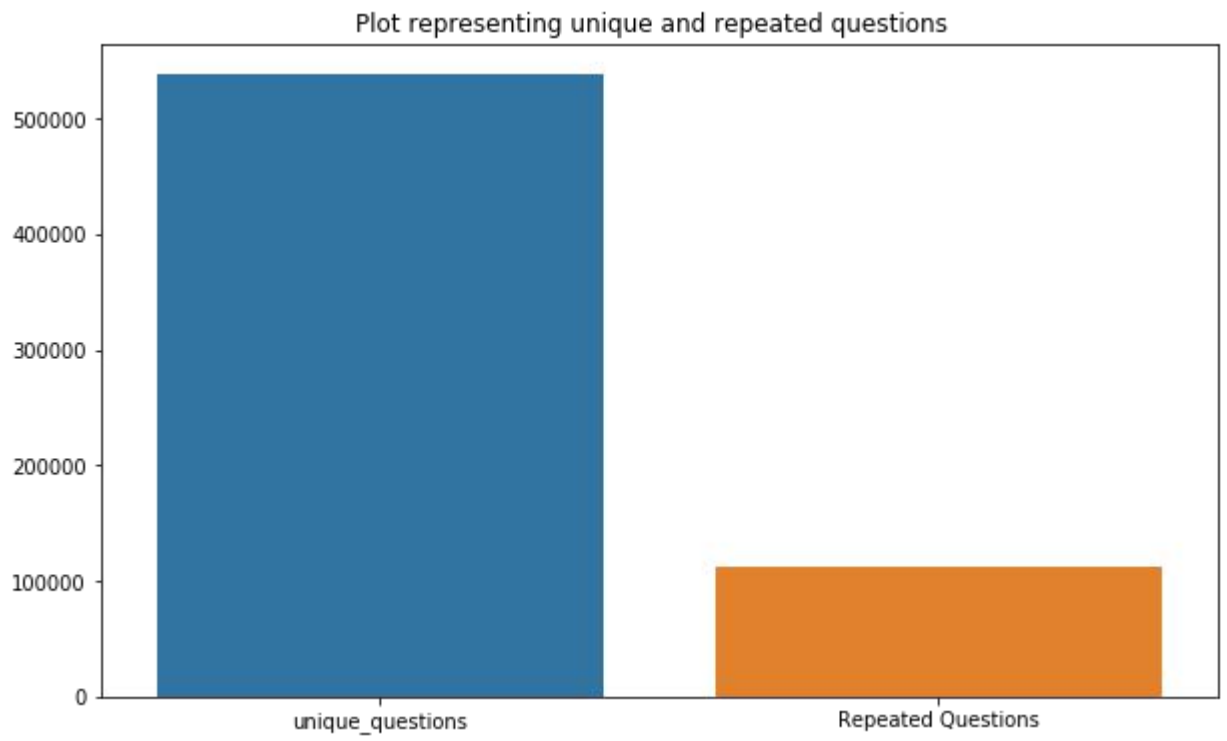
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.1 Number of unique questions

In [9]:
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(qs_morethan_onet
ime,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 [10]:
```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

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



3.2.2 Checking for Duplicates

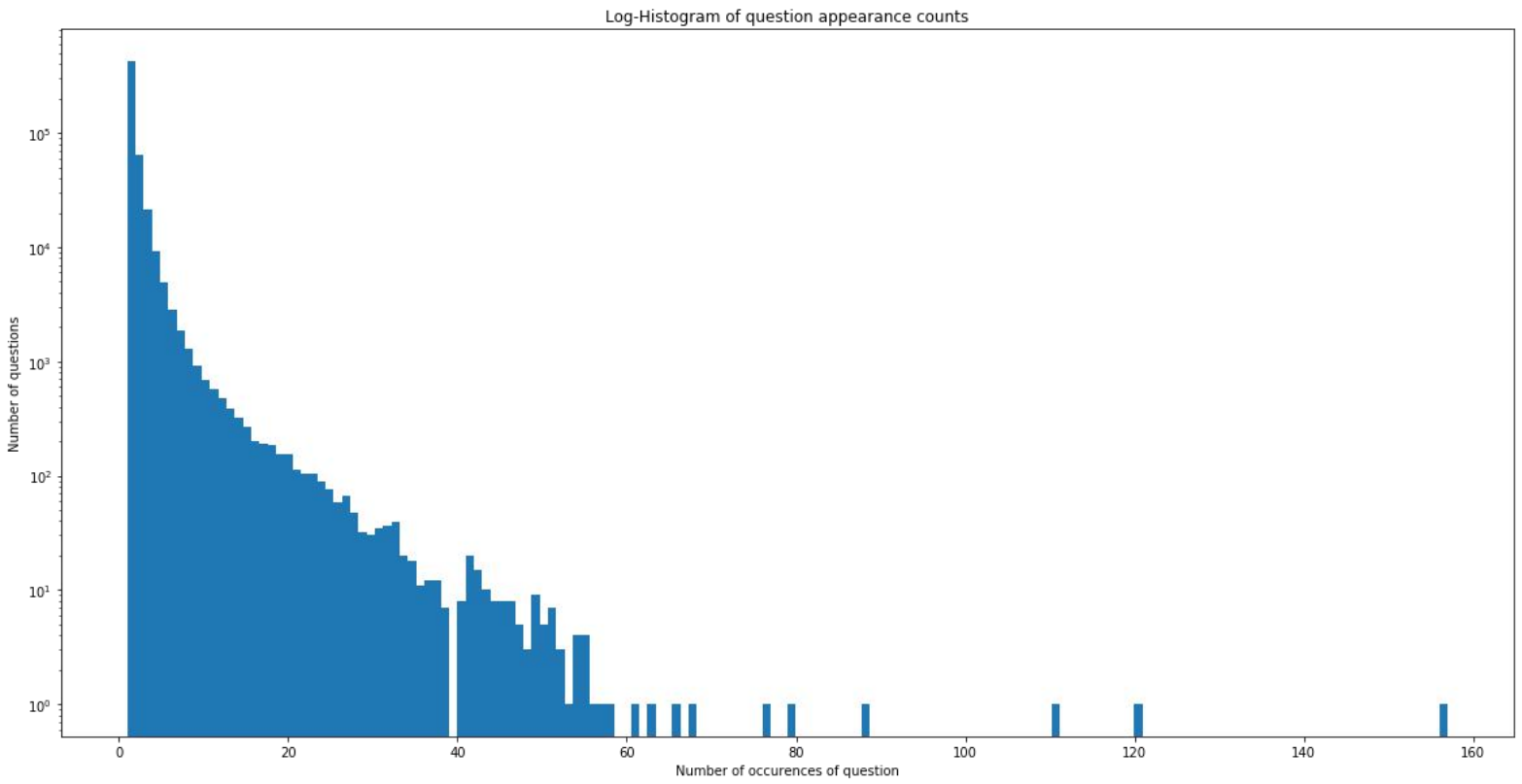
In [11]:
```
#checking whether there are any repeated pair of questions

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
```

3.2.3 Number of occurrences of each question

In [12]:
```
plt.figure(figsize=(20, 10))

plt.hist(qids.value_counts(), bins=160)

plt.yscale('log', nonposy='clip')

plt.title('Log-Histogram of question appearance counts')

plt.xlabel('Number of occurences of question')

plt.ylabel('Number of questions')

print ('Maximum number of times a single question is repeated: {}\n'.format(max(qids.value_counts())))
```

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



3.2.4 Checking for NULL values

In [13]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

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

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

- There are two rows with null values in question2

In [14]:
```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

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

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = 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 [15]:
```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

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

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

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

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

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

df.head()
```

Out[15]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
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
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
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
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
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

3.3.1 Analysis of some of the extracted features

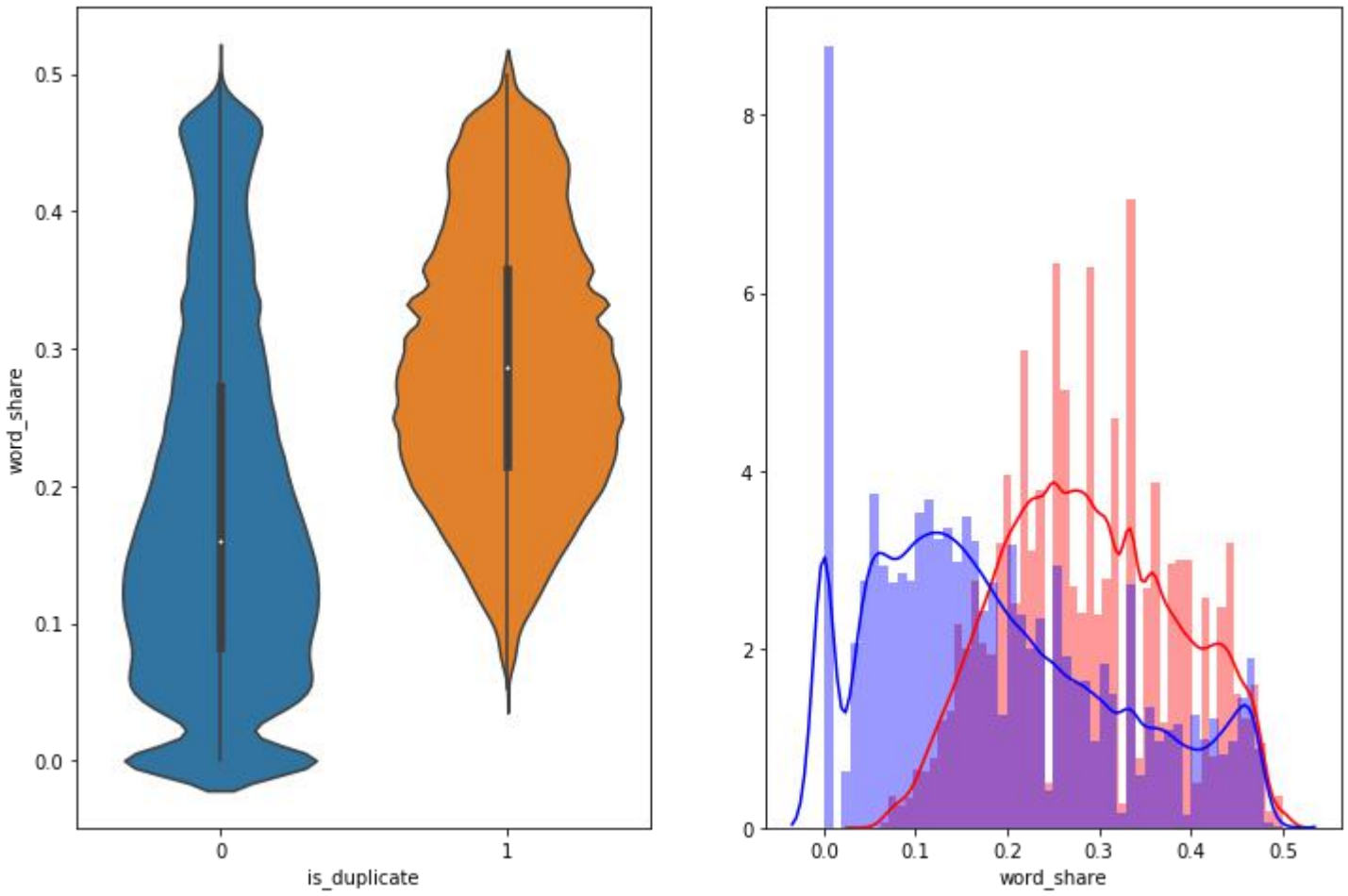
- Here are some questions have only one single words.

In [16]:
```
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

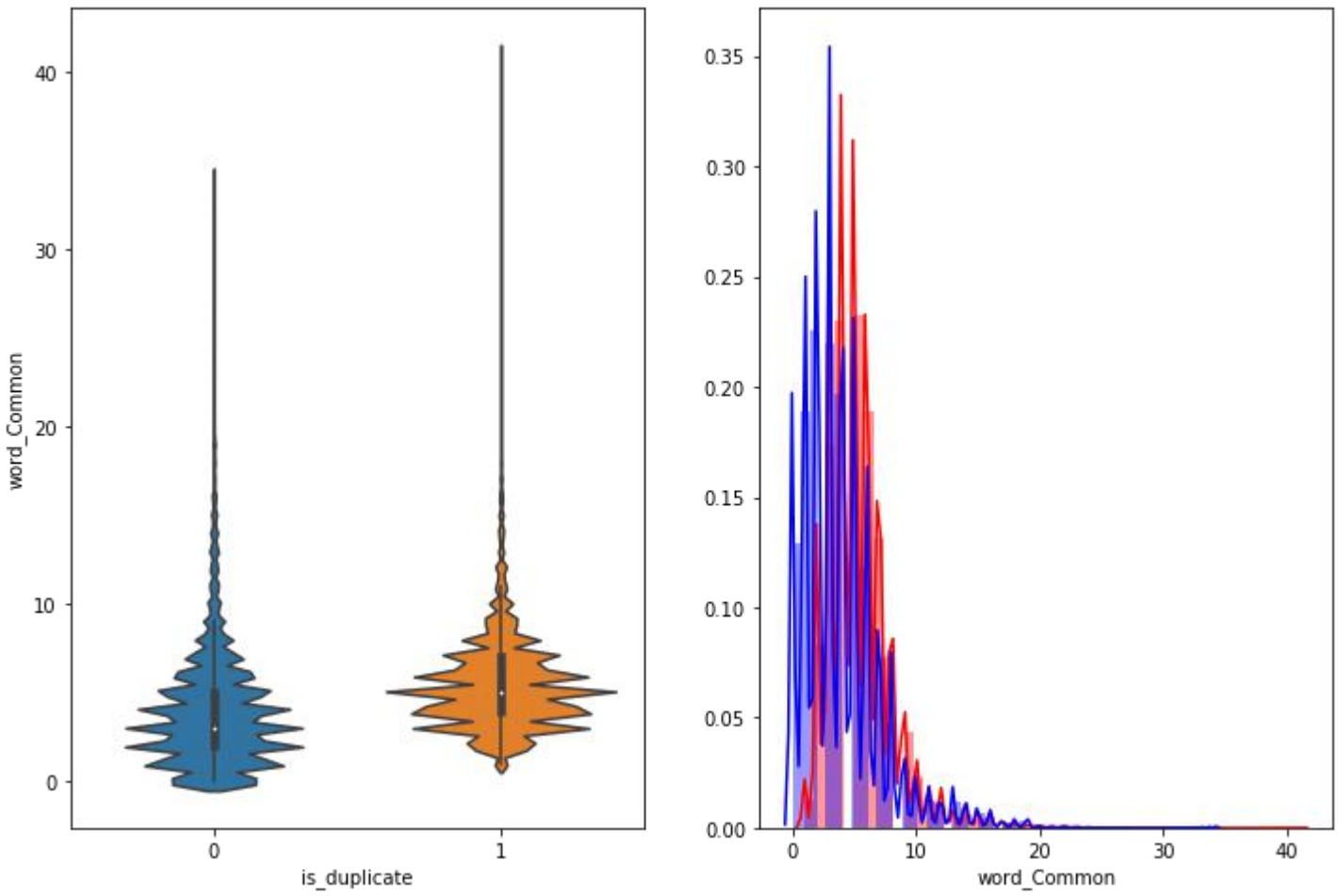
In [17]:
```
plt.figure(figsize=(12, 8))

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 [18]:
```
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

3.4 EDA: Advanced Feature Extraction.

In [19]:
```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byte-0x9c
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous notebook")
```

In [20]:
```
df.head(2)
```

Out[20]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
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
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

3.5 Preprocessing of Text

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

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

In [21]:
```
# To get the results in 4 decimal 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 i
s")\
                           .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
```

3.6 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

- **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 [22]:
```
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-string
s
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")
```

```
    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"], x["question
2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the tokens alphabeti
cally, 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["question
2"]), 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["quest
ion2"]), axis=1)
    return df
```

In [23]:
```
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[23]:

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

2 rows × 21 columns

3.7 Analysis of extracted features

3.7.1 Plotting Word clouds

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

In [24]:
```
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',encoding="utf-8")
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 [25]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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

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

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

Word Clouds generated from duplicate pair question's text

In [26]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.figure(figsize=(13,10))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

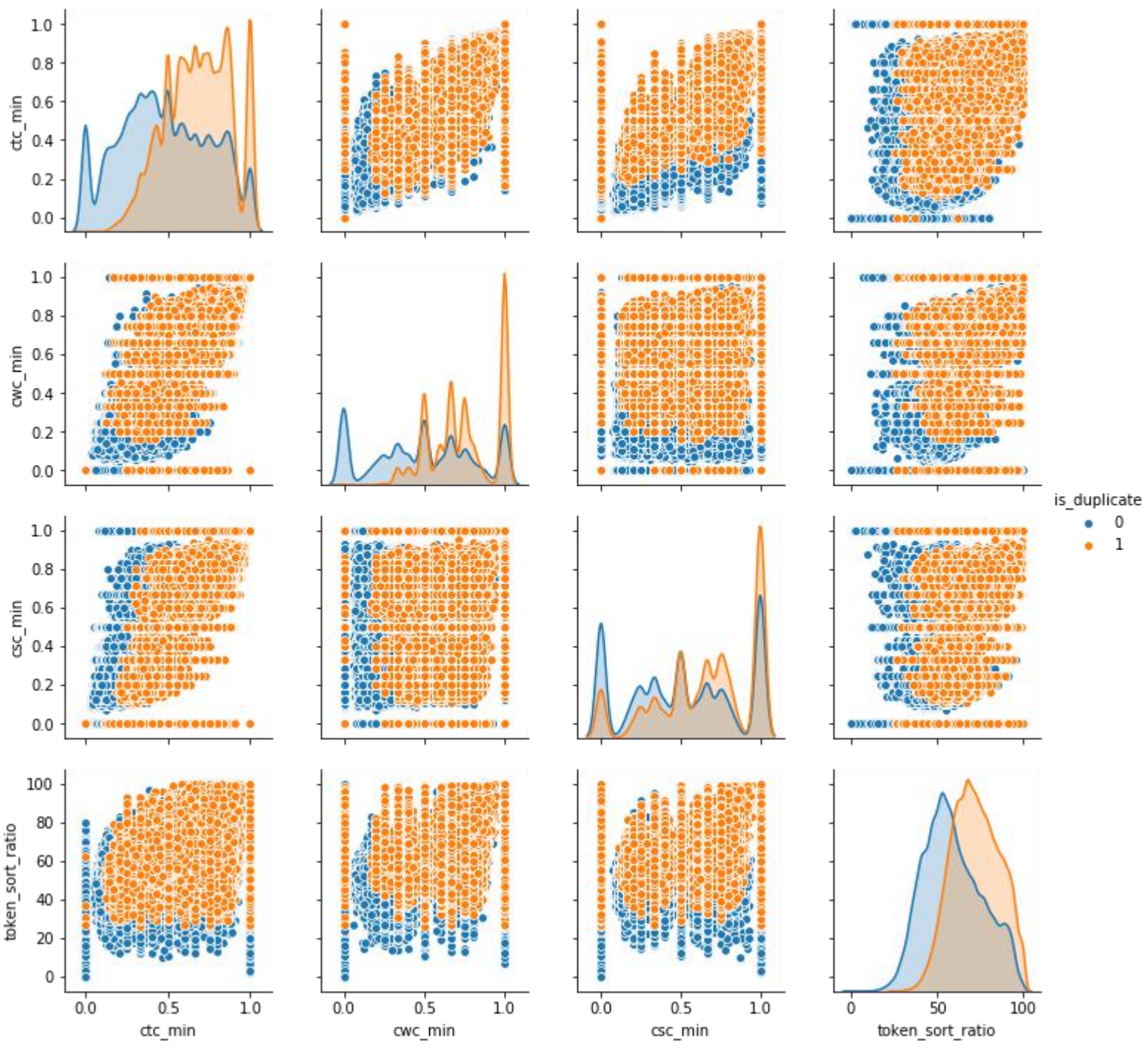
Word Clouds generated from non duplicate pair question's text

In [27]:
```
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.figure(figsize=(13,10))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

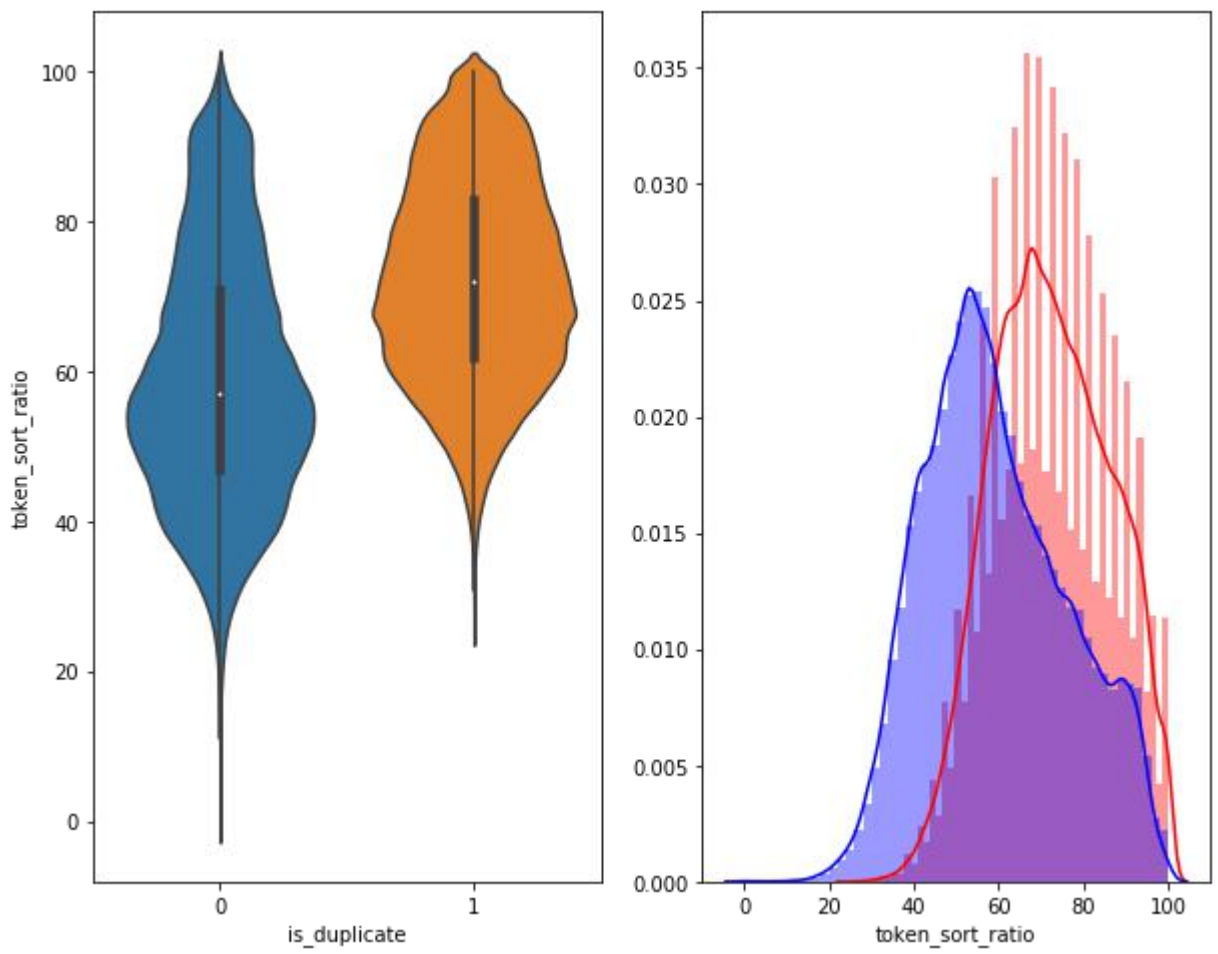
```
Word Cloud for non-Duplicate Question pairs:
```

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

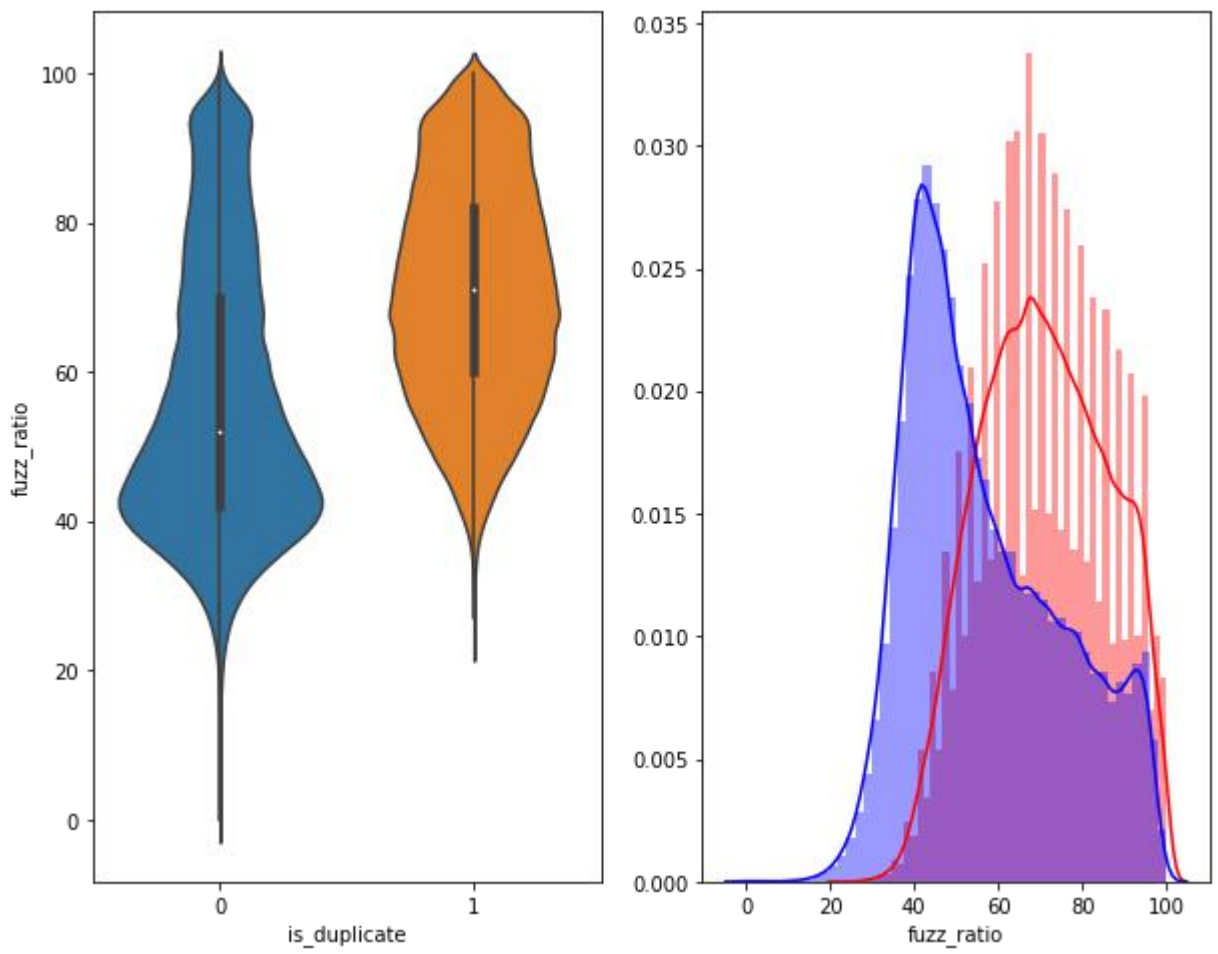
In [28]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min',
                 'cwc_min',
                 'csc_min',
                 'token_sort_ratio',
                 'is_duplicate']][0:n],
             hue='is_duplicate',
             vars=['ctc_min',
                   'cwc_min',
                   'csc_min',
                   'token_sort_ratio'])
plt.show()
```



In [29]:
```
# 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 [30]:
```
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()
```



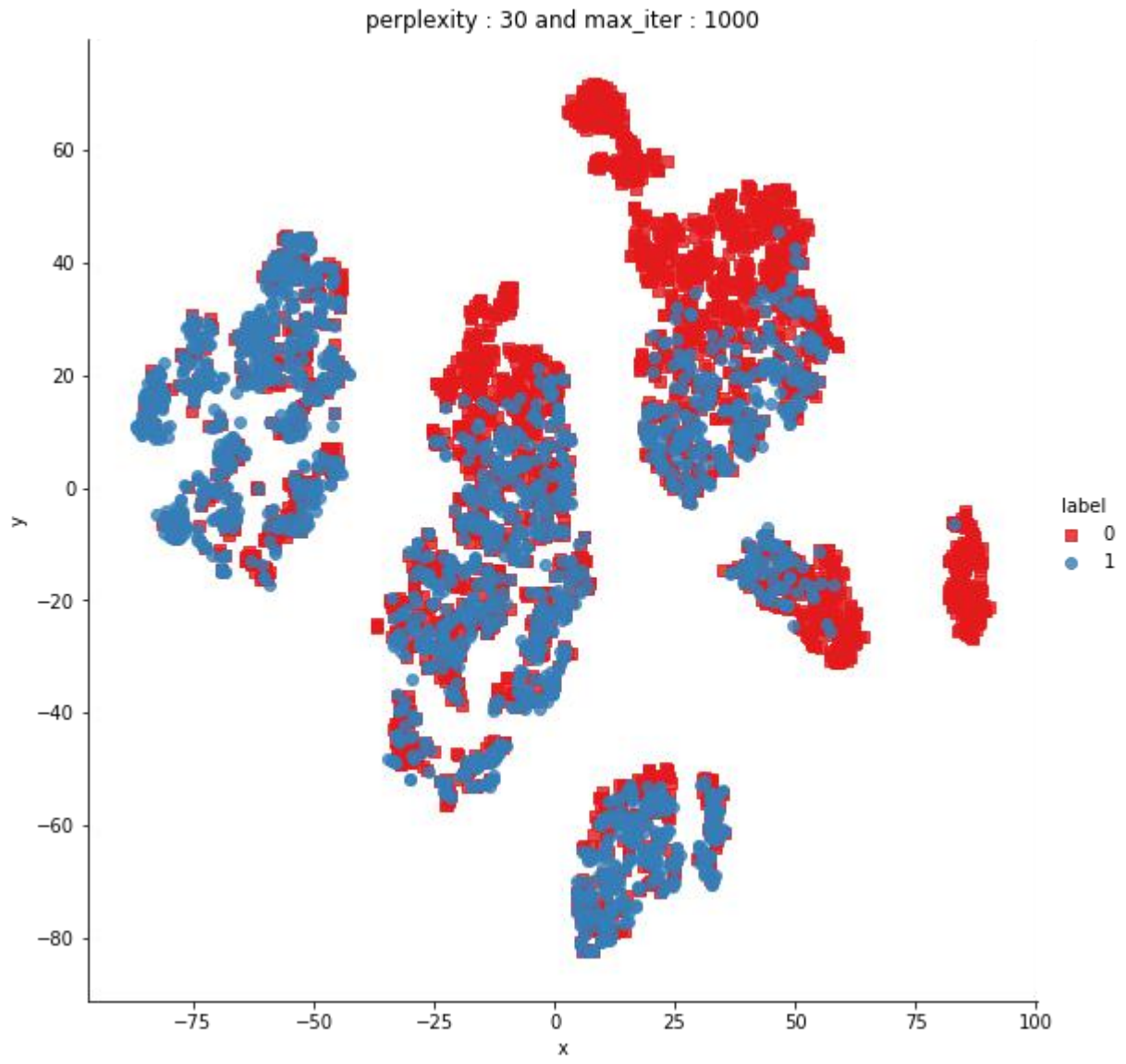
3.8 Dimensionality Reduction & Visualization

In [31]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 3 dime
ntion
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 [32]:
```
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 0.008s...
[t-SNE] Computed neighbors for 5000 samples in 0.296s...
[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 0.183s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iterations in 2.175s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iterations in 1.637s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iterations in 1.630s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iterations in 1.691s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iterations in 1.679s)
[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 1.683s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iterations in 1.628s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iterations in 1.647s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iterations in 1.624s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iterations in 1.677s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iterations in 1.656s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iterations in 1.655s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iterations in 1.689s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iterations in 1.659s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iterations in 1.672s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iterations in 1.673s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iterations in 1.677s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iterations in 1.650s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iterations in 1.699s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iterations in 1.686s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

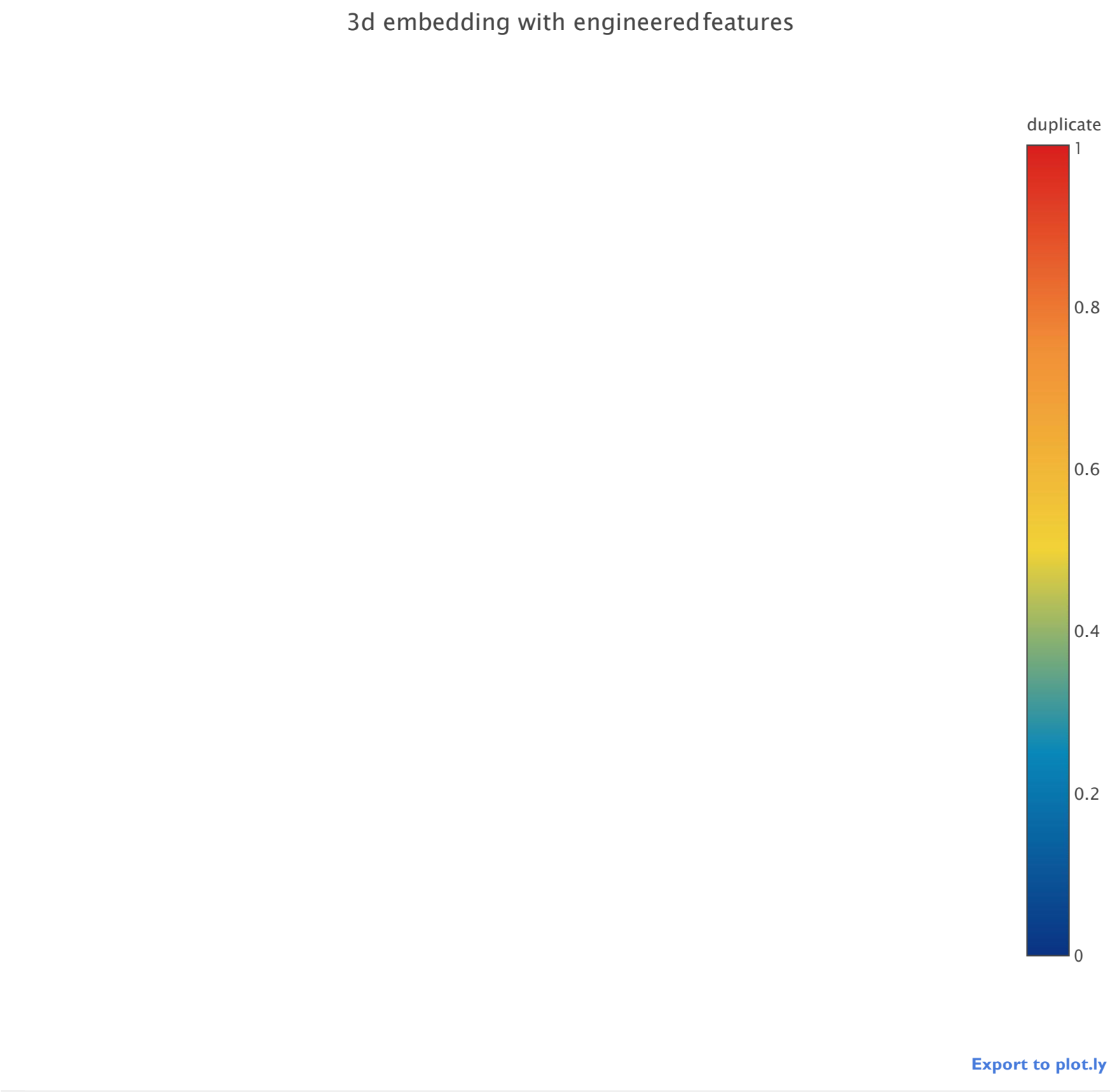
In [33]:
```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

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



In [34]:
```
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 0.306s...
[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 0.190s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 iterations in 7.755s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (50 iterations in 4.171s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (50 iterations in 3.706s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (50 iterations in 3.707s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (50 iterations in 3.663s)
[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 4.613s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50 iterations in 5.993s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50 iterations in 6.116s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50 iterations in 6.100s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50 iterations in 6.058s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50 iterations in 5.958s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50 iterations in 6.007s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50 iterations in 5.987s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50 iterations in 5.997s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50 iterations in 6.019s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50 iterations in 5.946s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50 iterations in 5.943s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50 iterations in 5.936s)
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50 iterations in 5.916s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (50 iterations in 6.021s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [39]:
```
       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)
```



4. Featurizing text data with tfidf weighted word-vectors

In [40]:
```
# avoid decoding problems
df = pd.read_csv("train.csv")

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

In [41]:
```
df.head()
```

Out[41]:

	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 [42]:
```
# merge texts
questions = list(df['question1']) + list(df['question2'])

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

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

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

In [43]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
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(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 384])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 404290/404290 [1:07:06<0
0:00, 100.40it/s]
```

In [44]:
```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 384])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 404290/404290 [1:09:35<
00:00, 96.82it/s]
```

In [45]:
```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('nlp_features_train.csv'):
    dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous notebook")

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [46]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)
df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
```

In [47]:
```
# dataframe of nlp features
df1.head()
```

Out[47]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_le
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0

In [48]:
```
# data before preprocessing
df2.head()
```

Out[48]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q
0	0	1	1	66	57	14	12	10.0	23.0	0.434783	2
1	1	4	1	51	88	8	13	4.0	20.0	0.200000	5
2	2	1	1	73	59	14	10	4.0	24.0	0.166667	2
3	3	1	1	50	65	11	9	0.0	19.0	0.000000	2
4	4	3	1	76	39	13	7	2.0	20.0	0.100000	4

In [49]:
```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[49]:

	0	1	2	3	4	5	6	7	8	
0	121.929927	100.083906	72.497900	115.641795	-48.370865	34.619070	-172.057790	-92.502626	113.223311	50.
1	-78.070935	54.843787	82.738495	98.191855	-51.234840	55.013509	-39.140733	-82.692374	45.161483	-9.
2	-5.355015	73.671810	14.376365	104.130241	1.433537	35.229116	-148.519385	-97.124595	41.972195	50.
3	5.778359	-34.712038	48.999631	59.699204	40.661263	-41.658731	-36.808594	24.170655	0.235601	-29
4	51.138220	38.587312	123.639488	53.333041	-47.062739	37.356212	-298.722753	-106.421119	106.248914	65.

5 rows × 384 columns

In [50]:
```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[50]:

	0	1	2	3	4	5	6	7	8	
0	125.983301	95.636484	42.114717	95.449986	-37.386301	39.400084	-148.116068	-87.851481	110.371972	62.2
1	-106.871899	80.290340	79.066300	59.302100	-42.175332	117.616657	-144.364242	-127.131506	22.962531	25.3
2	7.072875	15.513378	1.846914	85.937583	-33.808811	94.702337	-122.256856	-114.009530	53.922293	60.1
3	39.421539	44.136990	-24.010927	85.265864	-0.339028	-9.323141	-60.499653	-37.044767	49.407847	-23.3
4	31.950109	62.854102	1.778147	36.218763	-45.130861	66.674880	-106.342344	-22.901031	59.835921	62.6

5 rows × 384 columns

In [51]:
```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape[1])
print("Number of features in question2 w2v  dataframe :", df3_q2.shape[1])
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df3_q1.shape[1]+df3_q2.sha
pe[1])
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 384
Number of features in question2 w2v  dataframe : 384
Number of features in final dataframe  : 797
```

In [52]:
```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1['id']=df1['id']
    df3_q2['id']=df1['id']
    df1  = df1.merge(df2, on='id',how='left')
    df2  = df3_q1.merge(df3_q2, on='id',how='left')
    result  = df1.merge(df2, on='id',how='left')
    result.to_csv('final_features.csv')
```

5. Machine Learning Models

5.1 Reading data from file and storing into sql table

In [53]:
```
#Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('final_features.csv', names=['Unnamed: 0','id','is_duplicate','cwc_min','cwc
_max','csc_min','csc_max','ctc_min','ctc_max','last_word_eq','first_word_eq','abs_len_diff','mean_len'
,'token_set_ratio','token_sort_ratio','fuzz_ratio','fuzz_partial_ratio','longest_substr_ratio','freq_q
id1','freq_qid2','q1len','q2len','q1_n_words','q2_n_words','word_Common','word_Total','word_share','fr
eq_q1+q2','freq_q1-q2','0_x','1_x','2_x','3_x','4_x','5_x','6_x','7_x','8_x','9_x','10_x','11_x','12_
x','13_x','14_x','15_x','16_x','17_x','18_x','19_x','20_x','21_x','22_x','23_x','24_x','25_x','26_x',
'27_x','28_x','29_x','30_x','31_x','32_x','33_x','34_x','35_x','36_x','37_x','38_x','39_x','40_x','41_
x','42_x','43_x','44_x','45_x','46_x','47_x','48_x','49_x','50_x','51_x','52_x','53_x','54_x','55_x',
'56_x','57_x','58_x','59_x','60_x','61_x','62_x','63_x','64_x','65_x','66_x','67_x','68_x','69_x','70_
x','71_x','72_x','73_x','74_x','75_x','76_x','77_x','78_x','79_x','80_x','81_x','82_x','83_x','84_x',
'85_x','86_x','87_x','88_x','89_x','90_x','91_x','92_x','93_x','94_x','95_x','96_x','97_x','98_x','99_
x','100_x','101_x','102_x','103_x','104_x','105_x','106_x','107_x','108_x','109_x','110_x','111_x','11
2_x','113_x','114_x','115_x','116_x','117_x','118_x','119_x','120_x','121_x','122_x','123_x','124_x',
'125_x','126_x','127_x','128_x','129_x','130_x','131_x','132_x','133_x','134_x','135_x','136_x','137_
x','138_x','139_x','140_x','141_x','142_x','143_x','144_x','145_x','146_x','147_x','148_x','149_x','15
0_x','151_x','152_x','153_x','154_x','155_x','156_x','157_x','158_x','159_x','160_x','161_x','162_x',
'163_x','164_x','165_x','166_x','167_x','168_x','169_x','170_x','171_x','172_x','173_x','174_x','175_
x','176_x','177_x','178_x','179_x','180_x','181_x','182_x','183_x','184_x','185_x','186_x','187_x','18
8_x','189_x','190_x','191_x','192_x','193_x','194_x','195_x','196_x','197_x','198_x','199_x','200_x',
'201_x','202_x','203_x','204_x','205_x','206_x','207_x','208_x','209_x','210_x','211_x','212_x','213_
x','214_x','215_x','216_x','217_x','218_x','219_x','220_x','221_x','222_x','223_x','224_x','225_x','22
6_x','227_x','228_x','229_x','230_x','231_x','232_x','233_x','234_x','235_x','236_x','237_x','238_x',
'239_x','240_x','241_x','242_x','243_x','244_x','245_x','246_x','247_x','248_x','249_x','250_x','251_
x','252_x','253_x','254_x','255_x','256_x','257_x','258_x','259_x','260_x','261_x','262_x','263_x','26
4_x','265_x','266_x','267_x','268_x','269_x','270_x','271_x','272_x','273_x','274_x','275_x','276_x',
'277_x','278_x','279_x','280_x','281_x','282_x','283_x','284_x','285_x','286_x','287_x','288_x','289_
x','290_x','291_x','292_x','293_x','294_x','295_x','296_x','297_x','298_x','299_x','300_x','301_x','30
2_x','303_x','304_x','305_x','306_x','307_x','308_x','309_x','310_x','311_x','312_x','313_x','314_x',
'315_x','316_x','317_x','318_x','319_x','320_x','321_x','322_x','323_x','324_x','325_x','326_x','327_
x','328_x','329_x','330_x','331_x','332_x','333_x','334_x','335_x','336_x','337_x','338_x','339_x','34
0_x','341_x','342_x','343_x','344_x','345_x','346_x','347_x','348_x','349_x','350_x','351_x','352_x',
'353_x','354_x','355_x','356_x','357_x','358_x','359_x','360_x','361_x','362_x','363_x','364_x','365_
x','366_x','367_x','368_x','369_x','370_x','371_x','372_x','373_x','374_x','375_x','376_x','377_x','37
8_x','379_x','380_x','381_x','382_x','383_x','0_y','1_y','2_y','3_y','4_y','5_y','6_y','7_y','8_y','9_
y','10_y','11_y','12_y','13_y','14_y','15_y','16_y','17_y','18_y','19_y','20_y','21_y','22_y','23_y',
'24_y','25_y','26_y','27_y','28_y','29_y','30_y','31_y','32_y','33_y','34_y','35_y','36_y','37_y','38_
y','39_y','40_y','41_y','42_y','43_y','44_y','45_y','46_y','47_y','48_y','49_y','50_y','51_y','52_y',
'53_y','54_y','55_y','56_y','57_y','58_y','59_y','60_y','61_y','62_y','63_y','64_y','65_y','66_y','67_
y','68_y','69_y','70_y','71_y','72_y','73_y','74_y','75_y','76_y','77_y','78_y','79_y','80_y','81_y',
'82_y','83_y','84_y','85_y','86_y','87_y','88_y','89_y','90_y','91_y','92_y','93_y','94_y','95_y','96_
y','97_y','98_y','99_y','100_y','101_y','102_y','103_y','104_y','105_y','106_y','107_y','108_y','109_
y','110_y','111_y','112_y','113_y','114_y','115_y','116_y','117_y','118_y','119_y','120_y','121_y','12
2_y','123_y','124_y','125_y','126_y','127_y','128_y','129_y','130_y','131_y','132_y','133_y','134_y',
'135_y','136_y','137_y','138_y','139_y','140_y','141_y','142_y','143_y','144_y','145_y','146_y','147_
y','148_y','149_y','150_y','151_y','152_y','153_y','154_y','155_y','156_y','157_y','158_y','159_y','16
0_y','161_y','162_y','163_y','164_y','165_y','166_y','167_y','168_y','169_y','170_y','171_y','172_y',
'173_y','174_y','175_y','176_y','177_y','178_y','179_y','180_y','181_y','182_y','183_y','184_y','185_
y','186_y','187_y','188_y','189_y','190_y','191_y','192_y','193_y','194_y','195_y','196_y','197_y','19
8_y','199_y','200_y','201_y','202_y','203_y','204_y','205_y','206_y','207_y','208_y','209_y','210_y',
'211_y','212_y','213_y','214_y','215_y','216_y','217_y','218_y','219_y','220_y','221_y','222_y','223_
y','224_y','225_y','226_y','227_y','228_y','229_y','230_y','231_y','232_y','233_y','234_y','235_y','23
6_y','237_y','238_y','239_y','240_y','241_y','242_y','243_y','244_y','245_y','246_y','247_y','248_y',
'249_y','250_y','251_y','252_y','253_y','254_y','255_y','256_y','257_y','258_y','259_y','260_y','261_
y','262_y','263_y','264_y','265_y','266_y','267_y','268_y','269_y','270_y','271_y','272_y','273_y','27
4_y','275_y','276_y','277_y','278_y','279_y','280_y','281_y','282_y','283_y','284_y','285_y','286_y',
'287_y','288_y','289_y','290_y','291_y','292_y','293_y','294_y','295_y','296_y','297_y','298_y','299_
y','300_y','301_y','302_y','303_y','304_y','305_y','306_y','307_y','308_y','309_y','310_y','311_y','31
2_y','313_y','314_y','315_y','316_y','317_y','318_y','319_y','320_y','321_y','322_y','323_y','324_y',
'325_y','326_y','327_y','328_y','329_y','330_y','331_y','332_y','333_y','334_y','335_y','336_y','337_
y','338_y','339_y','340_y','341_y','342_y','343_y','344_y','345_y','346_y','347_y','348_y','349_y','35
0_y','351_y','352_y','353_y','354_y','355_y','356_y','357_y','358_y','359_y','360_y','361_y','362_y',
'363_y','364_y','365_y','366_y','367_y','368_y','369_y','370_y','371_y','372_y','373_y','374_y','375_
y','376_y','377_y','378_y','379_y','380_y','381_y','382_y','383_y'], chunksize=chunksize, iterator=Tru
e, encoding='utf-8', ):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
```

In [54]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None


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

In [55]:
```
read_db = 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
data
```

In [56]:
```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)

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

In [57]:
```
# remove the first row
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace=True)
```

In [58]:
```
data.head()
```

Out[58]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	c
1	0.749981250468738	0.599988000239995	0.999980000399992	0.999980000399992	0.888879012455417	0.727266115
2	0.999950002499875	0.66664444518516	0.999966667777741	0.999966667777741	0.999980000399992	0.833319444
3	0.999966667777741	0.749981250468738	0.999975000624984	0.999975000624984	0.874989062636717	0.874989062
4	0.33332222259258	0.249993750156246	0.0	0.0	0.14285510206997	0.124998437
5	0.66664444518516	0.66664444518516	0.999983333611106	0.857130612419823	0.888879012455417	0.799992000

5 rows × 794 columns

5.2 Converting strings to numerics

In [60]:
```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
cols = list(data.columns)
data = pd.DataFrame(np.array(data.values,dtype=np.float64),columns=cols)
```

In [61]:
```
data.head()
```

Out[61]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len
0	0.749981	0.599988	0.999980	0.999980	0.888879	0.727266	0.0	1.0	2.0	10.0
1	0.999950	0.666644	0.999967	0.999967	0.999980	0.833319	0.0	1.0	1.0	5.5
2	0.999967	0.749981	0.999975	0.999975	0.874989	0.874989	1.0	1.0	0.0	8.0
3	0.333322	0.249994	0.000000	0.000000	0.142855	0.124998	0.0	0.0	1.0	7.5
4	0.666644	0.666644	0.999983	0.857131	0.888879	0.799992	0.0	1.0	1.0	9.5

5 rows × 794 columns

In [62]:
```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
y_true = list(map(int, y_true.values))
```

5.3 Random train test split(70:30)

In [63]:
```
X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

In [64]:
```
print("Number of data points in train data :",X_train.shape)
print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (70000, 794)
Number of data points in test data : (30000, 794)
```

In [65]:
```
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)
```

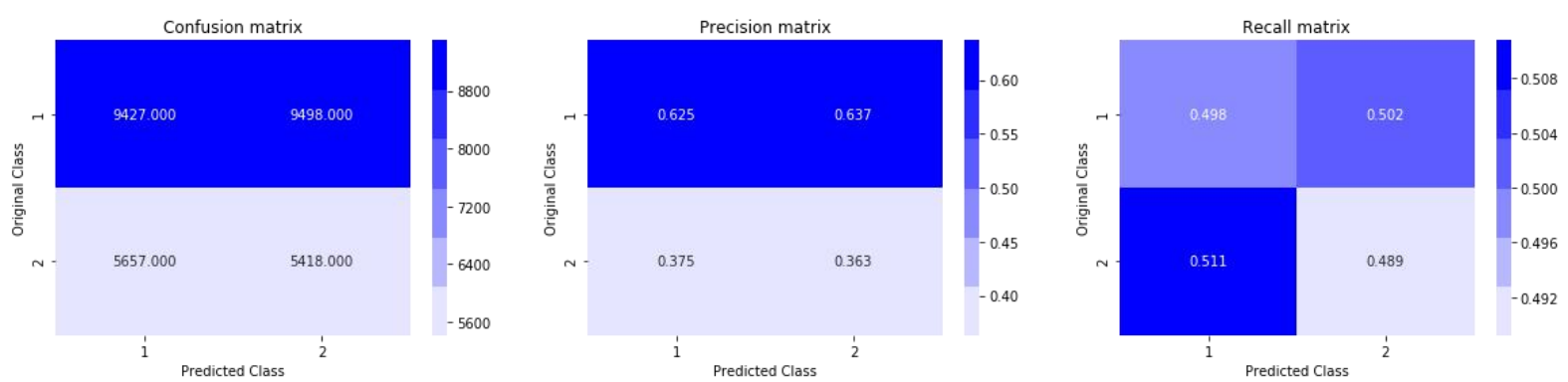
```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308285714285714 Class 1:  0.3691714285714286
---------- Distribution of output variable in train data ----------
Class 0:  0.36916666666666664 Class 1:  0.36916666666666664
```

In [66]:
```
# 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()
```

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

In [67]:
```
# 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 0.8908775300448231
```

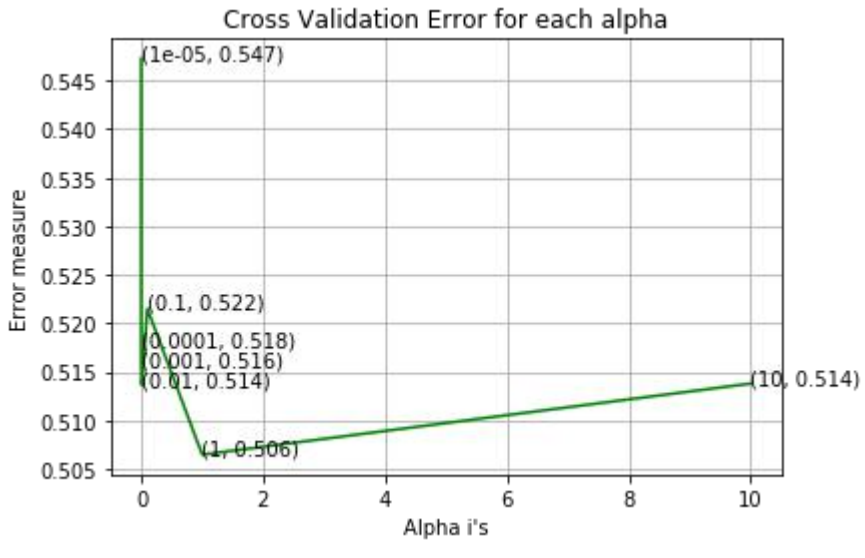


5.5 Logistic Regression with hyperparameter tuning

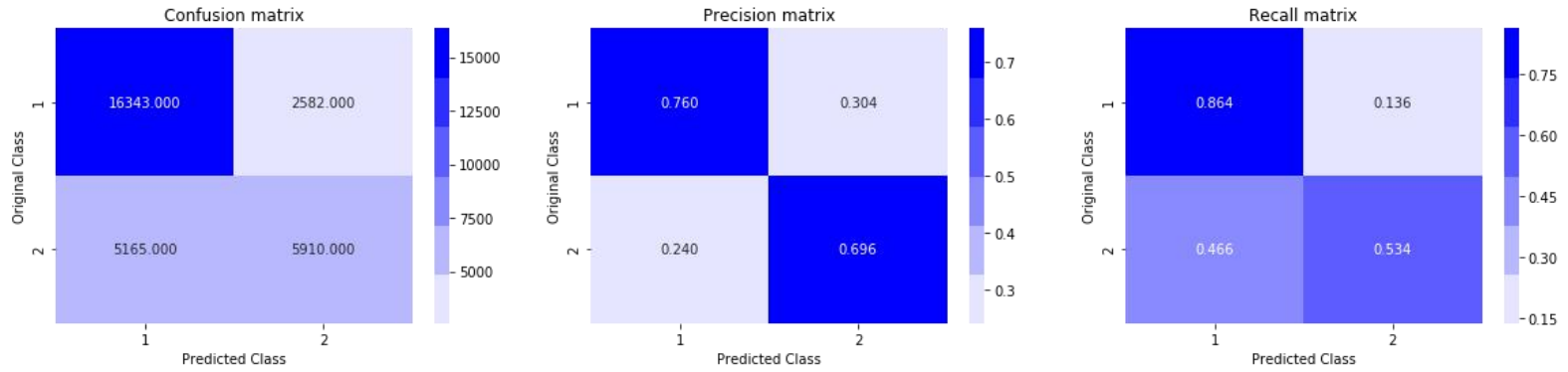
In [68]:

```
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.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='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=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: 0.5472666260767256
For values of alpha =  0.0001 The log loss is: 0.5175839987451992
For values of alpha =  0.001 The log loss is: 0.5156285311252452
For values of alpha =  0.01 The log loss is: 0.5136506329178379
For values of alpha =  0.1 The log loss is: 0.5215090248298853
For values of alpha =  1 The log loss is: 0.5064837390256053
For values of alpha =  10 The log loss is: 0.5138003177631272
```



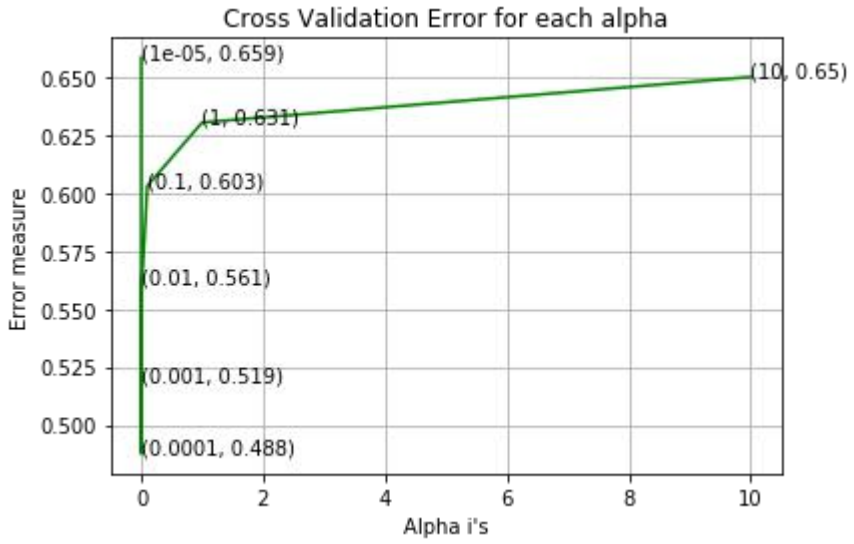
```
For values of best alpha =  1 The train log loss is: 0.5009828437665361
For values of best alpha =  1 The test log loss is: 0.5064837390256053
Total number of data points : 30000
```



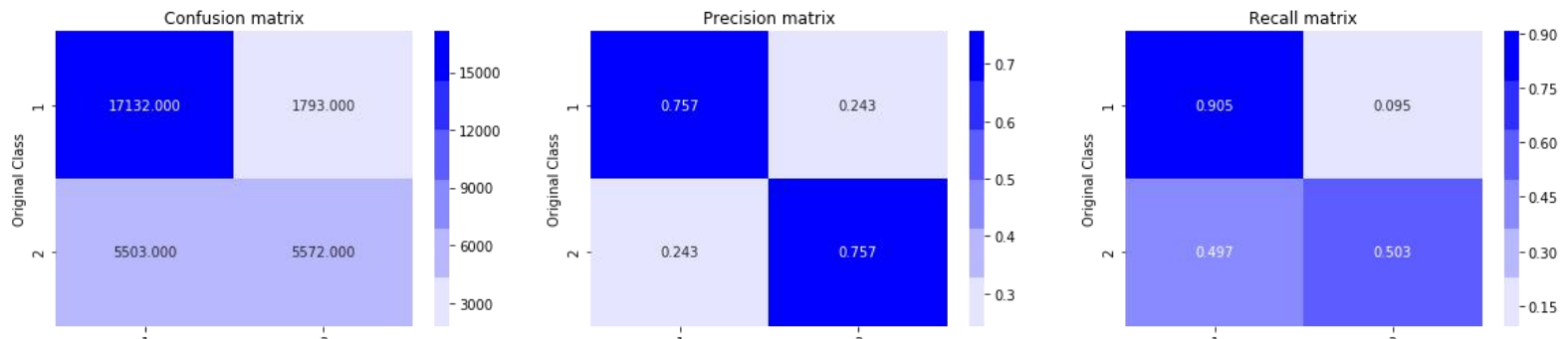
5.6 Linear SVM with hyperparameter tuning

In [69]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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=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: 0.6585106771722132
For values of alpha =  0.0001 The log loss is: 0.48759698525732137
For values of alpha =  0.001 The log loss is: 0.5190034189962978
For values of alpha =  0.01 The log loss is: 0.5609078071961177
For values of alpha =  0.1 The log loss is: 0.603200984412773
For values of alpha =  1 The log loss is: 0.6305138658937702
For values of alpha =  10 The log loss is: 0.6501659120491334
```




```
For values of best alpha = 0.0001 The train log loss is: 0.4797110523956667
For values of best alpha = 0.0001 The test log loss is: 0.48759698525732137
Total number of data points : 30000
```



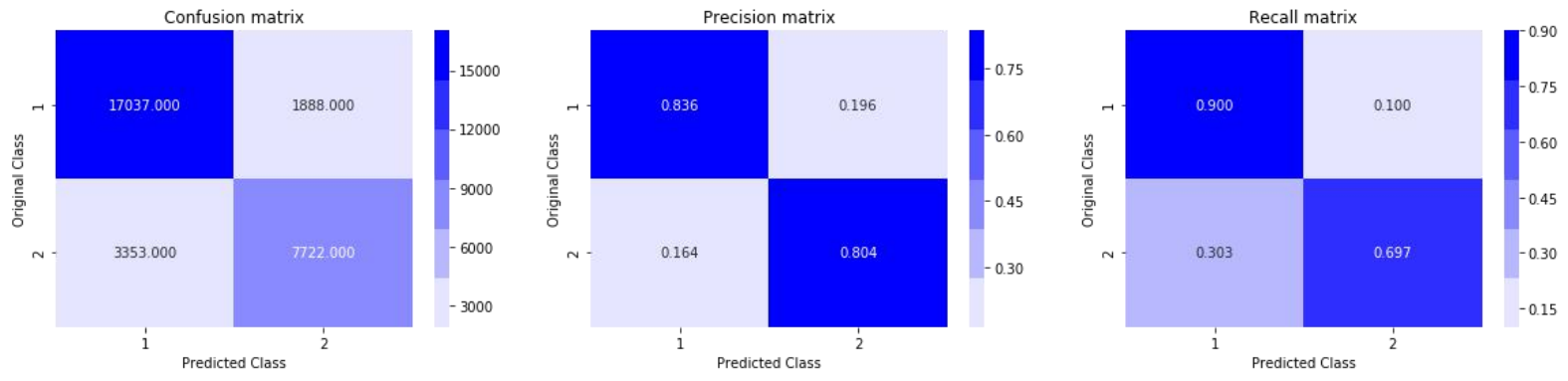
5.7 XGBoost Model

In [83]:
```
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4
params['silent'] = 1

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,verbose_eval= False,early_stopping_rounds=20)

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))
```

```
The test log loss is: 0.3576330536834139
```

In [84]:
```
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
```



5.7 Perfom Modeling on complete dataset with TF-IDF Features

In [174]:
```
# Load Basic Features
df_basic_feature = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
```

In [175]:
```
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_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')

Number of columns :  17
```

Out[175]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
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
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
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
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
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

In [176]:
```
# Load Advance Features
df_advance_features = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
```

In [177]:
```
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_duplicate',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')

Number of columns :  21
```

Out[177]:

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

5 rows × 21 columns

Lets drop 'qid1', 'qid2','question1','question2','is_duplicate' columns and add both the dataframe

In [178]:
```
# 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','question2','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='id',how='left')
```

Lets check for NaN values

In [179]:
```
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]
```

Found 3 such rows, we will remove these rows from the original dataset itself

In [180]:
```
df_basic_advance_features = df_basic_advance_features[df_basic_advance_features['question1'].notnull
()]
df_basic_advance_features = df_basic_advance_features[df_basic_advance_features['question2'].notnull
()]
```

In [181]:
```
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, 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_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]
Index: []

[0 rows x 30 columns]
```

In [182]:
```
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 [183]:
```
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_qid1',
       '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[183]:

	id	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	...	ctc_m
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	12	...	0.7857
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	13	...	0.4666
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	10	...	0.2857
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	9	...	0.0000
4	4	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	13	7	...	0.3076

5 rows × 30 columns

Lets separate the target feature

In [184]:
```
target = df_basic_advance_features['is_duplicate']
```

Also drop the unnecessary id and is_duplicate column from feature dataset

In [185]:
```
df_basic_advance_features.drop(['id','is_duplicate'], axis=1, inplace=True)
```

In [186]:
```
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', '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 :  28
```

Out[186]:

	question1	question2	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Tota
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	10.0	23.0
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	4.0	20.0
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	4.0	24.0
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	0.0	19.0
4	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	3	1	76	39	13	7	2.0	20.0

5 rows × 28 columns

5.7.1 Perform TF-IDF Tokenization on columns- 'question1', 'question2'

In [187]:
```
# Instanciate Tfidf Vectorizer
tfidfVectorizer_question1 = TfidfVectorizer()

question1_dtm = tfidfVectorizer_question1.fit_transform(df_basic_advance_features['question1'].values.
astype('U'))
```

In [188]:
```
print("Found {0} features from question1 column".format(len(tfidfVectorizer_question1.get_feature_name
s())))
```

```
Found 67525 features from question1 column
```

In [189]:
```
# Instanciate Tfidf Vectorizer
tfidfVectorizer_question2 = TfidfVectorizer()

question2_dtm = tfidfVectorizer_question2.fit_transform(df_basic_advance_features['question2'].values.
astype('U'))
```

In [190]:
```
print("Found {0} features from question2 column".format(len(tfidfVectorizer_question2.get_feature_name
s())))
```

```
Found 62331 features from question2 column
```

In [191]:
```
# Combine all the features in question1 and question2
question1_question2 = hstack((question1_dtm,question2_dtm))
```

In [195]:
```
type(question1_question2)
```

Out[195]:
```
scipy.sparse.coo.coo_matrix
```

In [196]:
```
# Drop unnecessary question1 and question2 columns
df_basic_advance_features.drop(['question1','question2'], axis=1, inplace=True)
```

In [197]:
```
# Combine all basic, advance and tfidf features
df_basic_advance_tfidf_features = hstack((df_basic_advance_features, question1_question2),format="csr"
,dtype='float64')
```

In [199]:
```
df_basic_advance_tfidf_features.shape
```

Out[199]: (404287, 129882)

5.7.2 Split data into 70:30

In [203]:
```
x_train,x_test, y_train, y_test = train_test_split(df_basic_advance_tfidf_features, target, stratify=t
arget, test_size=0.3)
```

In [204]:
```
print("Number of data points in train data :",x_train.shape)
print("Number of data points in test data :",x_test.shape)
```

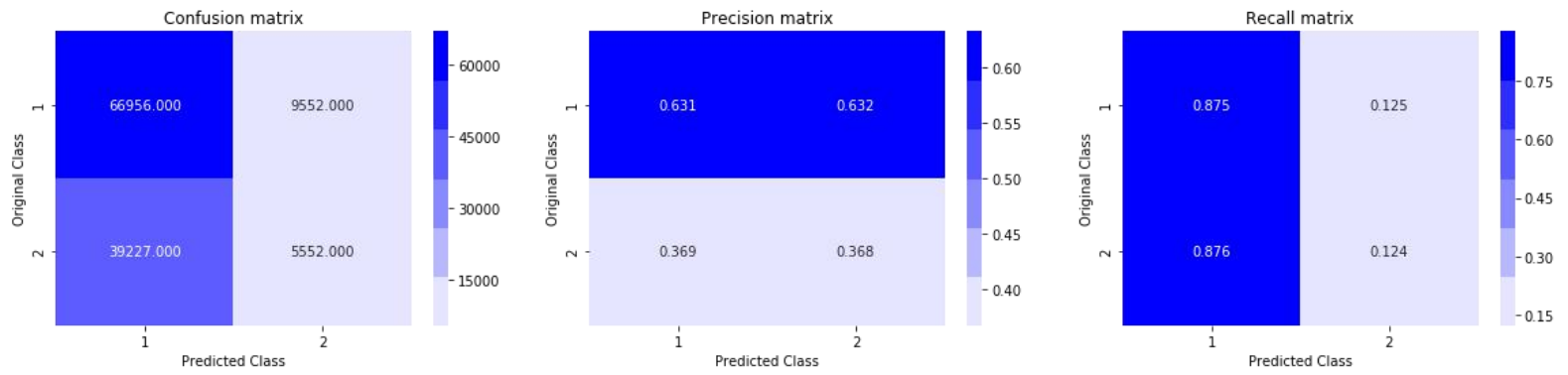
```
Number of data points in train data : (283000, 129882)
Number of data points in test data : (121287, 129882)
```

5.8 Apply ML Models

5.8.1 Random Model

In [207]:
```
predicted_y = np.zeros((len(y_test),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)
```

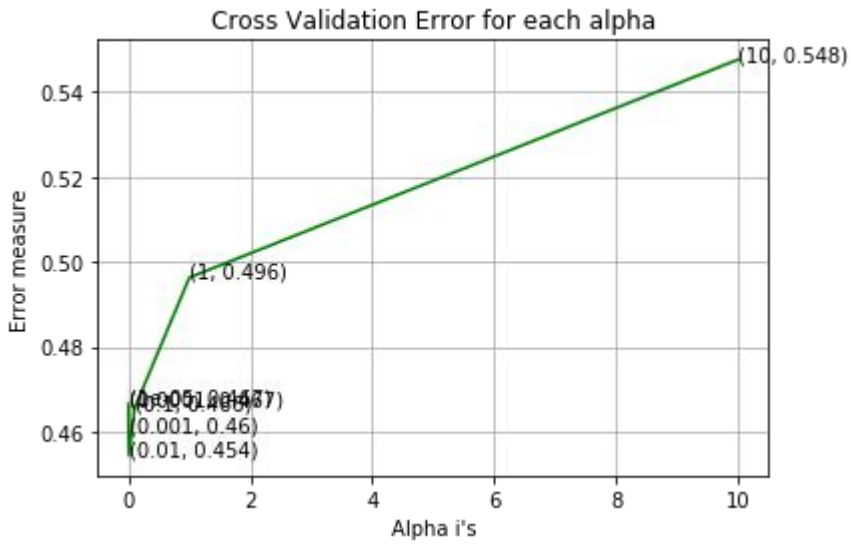
```
Log loss on Test Data using Random Model 0.7406965895681863
```



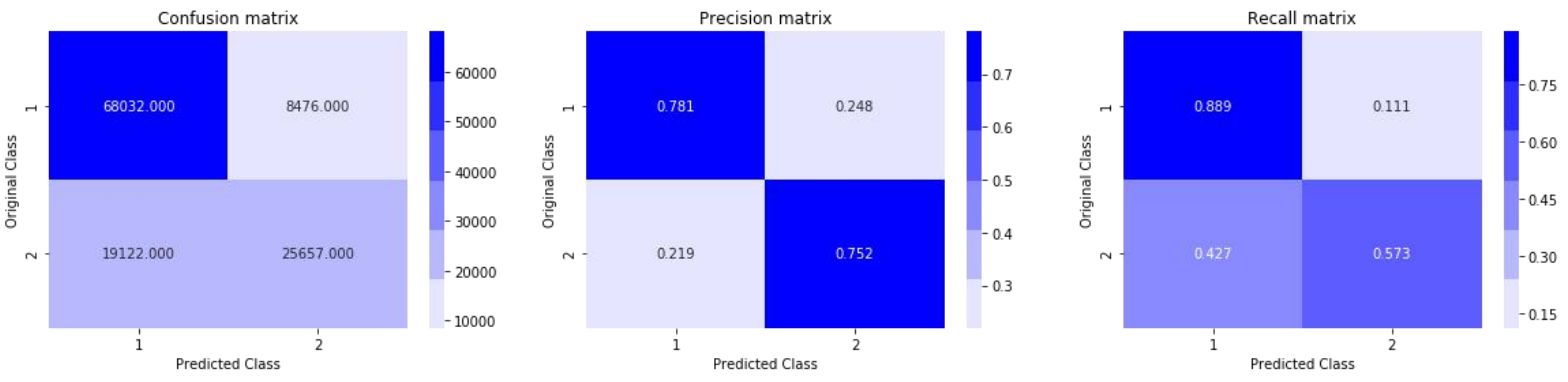
5.8.2 Logistic Regression with hyperparameter tuning

In [208]:
```
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=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.class
es_, 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, pre
dict_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, predi
ct_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: 0.46661572033158716
For values of alpha =  0.0001 The log loss is: 0.4665460514852834
For values of alpha =  0.001 The log loss is: 0.46026542309356433
For values of alpha =  0.01 The log loss is: 0.4543964333876638
For values of alpha =  0.1 The log loss is: 0.46560038927189823
For values of alpha =  1 The log loss is: 0.49633976893385323
For values of alpha =  10 The log loss is: 0.5476579656462788
```



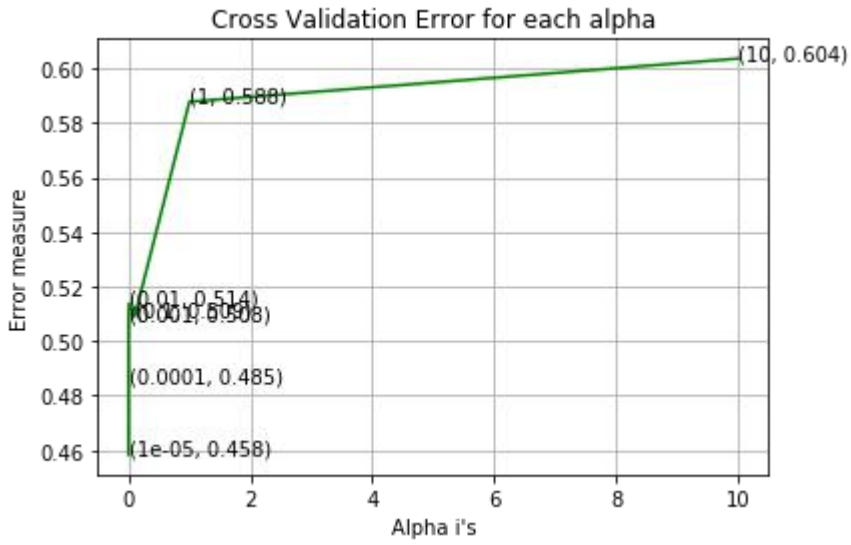

```
For values of best alpha =  0.01 The train log loss is: 0.4549658356952683
For values of best alpha =  0.01 The test log loss is: 0.4543964333876638
Total number of data points : 121287
```



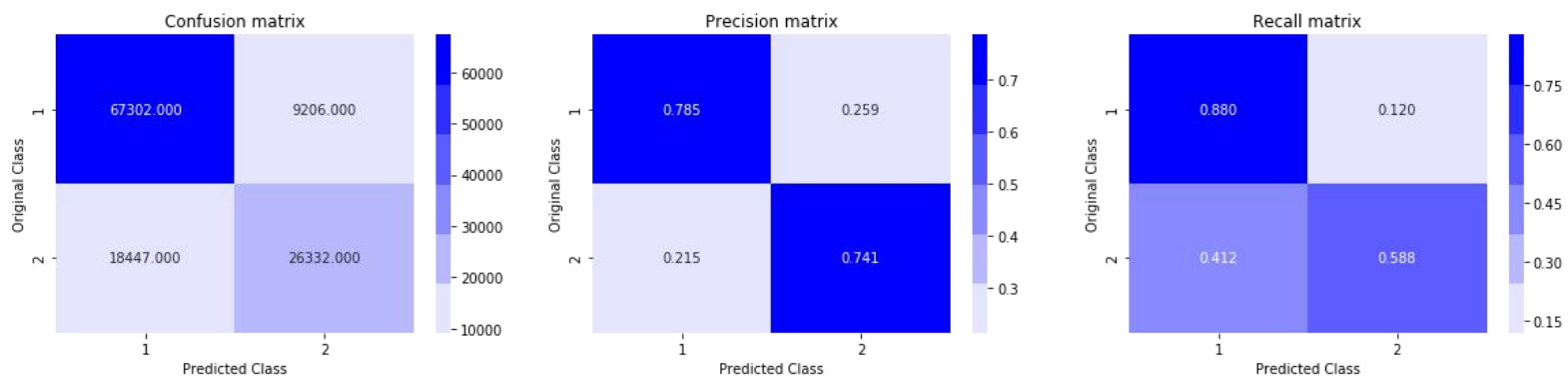
5.8.3 Linear SVM with hyperparameter tuning

In [209]:
```
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=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.class
es_, 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, pre
dict_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, predi
ct_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: 0.4583474012307617
For values of alpha =  0.0001 The log loss is: 0.4852737096283473
For values of alpha =  0.001 The log loss is: 0.5078547515068808
For values of alpha =  0.01 The log loss is: 0.5137362023362886
For values of alpha =  0.1 The log loss is: 0.5089026847606254
For values of alpha =  1 The log loss is: 0.5877475161723162
For values of alpha =  10 The log loss is: 0.6035517972721499
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4594298922386731
For values of best alpha =  1e-05 The test log loss is: 0.4583474012307617
Total number of data points : 121287
```



5.8.4 XGBoost Model

In [217]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV,StratifiedKFold
```

In [251]:
```
print("Feature Shape: ",data.shape)
print("Target Shape: ",len(y_true))
```

```
Feature Shape:  (100000, 794)
Target Shape:  100000
```

In [218]:
```
feature_train, feature_test, target_train, target_test = train_test_split(data, y_true, stratify=y_tru
e, test_size=0.3)
```

In [219]:
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(target_train)
train_len = len(target_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(target_test)
test_len = len(target_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:  0.6308285714285714 Class 1:  0.3691714285714286
---------- Distribution of output variable in train data ----------
Class 0:  0.36916666666666664 Class 1:  0.36916666666666664
```

In [243]:
```
n_estimators = [100, 300, 500, 700, 900, 1100, 1300, 1500]
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
colsample_bytree = [0.1, 0.3, 0.5, 0.7 , 0.9, 1]
subsample = [0.1,0.3,0.5,0.7,0.9,1]

def hyperparameter_tunning(X,Y):
    param_grid = dict(learning_rate=learning_rate,
                      n_estimators=n_estimators,
                      colsample_bytree = colsample_bytree,
                      subsample = subsample)

    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.best_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 [241]:
```
start = dt.datetime.now()

# Tune hyperparameter values
random_result = hyperparameter_tunning(feature_train,target_train)

print("\nTimeTaken: ",dt.datetime.now() - start)
```

```
Best: -0.348366 using {'subsample': 0.3, 'n_estimators': 1100, 'learning_rate': 0.1, 'colsample_bytre
e': 0.7}

-0.368705 (0.004740) with: {'subsample': 0.7, 'n_estimators': 900, 'learning_rate': 0.3, 'colsample_by
tree': 0.3}
-0.474010 (0.002540) with: {'subsample': 1, 'n_estimators': 1300, 'learning_rate': 0.001, 'colsample_b
ytree': 0.5}
-0.663301 (0.000289) with: {'subsample': 0.5, 'n_estimators': 900, 'learning_rate': 0.0001, 'colsample
_bytree': 0.5}
-0.364430 (0.004894) with: {'subsample': 0.1, 'n_estimators': 300, 'learning_rate': 0.1, 'colsample_by
tree': 1}
-0.560023 (0.001351) with: {'subsample': 1, 'n_estimators': 500, 'learning_rate': 0.001, 'colsample_by
tree': 1}
nan (nan) with: {'subsample': 0.1, 'n_estimators': 1500, 'learning_rate': 0.2, 'colsample_bytree': 0.
5}
-0.402128 (0.004055) with: {'subsample': 0.1, 'n_estimators': 700, 'learning_rate': 0.01, 'colsample_b
ytree': 0.1}
-0.348366 (0.004479) with: {'subsample': 0.3, 'n_estimators': 1100, 'learning_rate': 0.1, 'colsample_b
ytree': 0.7}
-0.388525 (0.003834) with: {'subsample': 0.7, 'n_estimators': 1500, 'learning_rate': 0.3, 'colsample_b
ytree': 0.5}
-0.453984 (0.002640) with: {'subsample': 0.5, 'n_estimators': 1500, 'learning_rate': 0.001, 'colsample
_bytree': 1}

TimeTaken:  3:10:28.435725
```

In [246]:
```
xGBClassifier = XGBClassifier(max_depth=3,
                              learning_rate=0.1,
                              n_estimators=1100,
                              subsample=0.3,
                              colsample_bytree= 0.7,
                              nthread=-1)
xGBClassifier
```

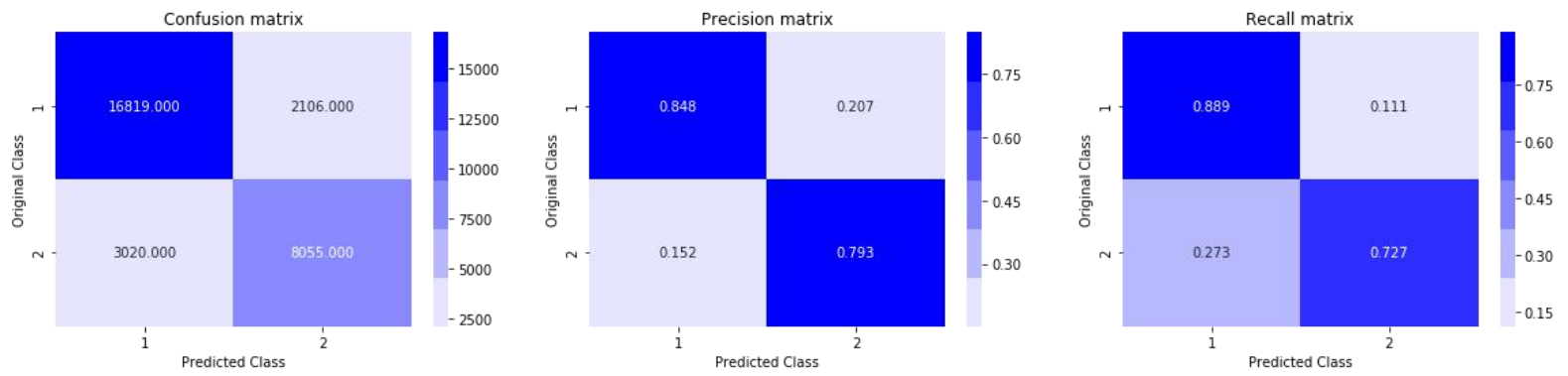
Out[246]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.7, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=1100,
       n_jobs=1, nthread=-1, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.3)
```

In [254]:
```
start = dt.datetime.now()
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 3
params['colsample_bytree'] = 0.7
params['n_estimators'] = 1100
params['subsample'] = 0.3
params['learning_rate'] = 0.1
params['nthread'] = -1
params['silent'] = 1

d_train = xgb.DMatrix(feature_train, label=target_train)
d_test = xgb.DMatrix(feature_test, label=target_test)

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

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

xgdmat = xgb.DMatrix(feature_train,target_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(target_test, predict_y, labels=clf.classes_, eps=1e-15))
print("\nTime Taken: ",dt.datetime.now() - start)
```

```
The test log loss is: 0.3482895103792922

Time Taken:  0:08:43.361790
```

In [256]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(target_test, predicted_y)
```

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



Conclusion

In [1]:
```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ['Dataset Size', 'Model Name', 'Tokenizer','Hyperparameter Tunning', 'Test Log Lo
ss']
ptable.add_row(["~ 100K","Random","TFIDF Weighted W2V","NA","0.89"])
ptable.add_row(["~ 100K","Logistic Regression","TFIDF Weighted W2V","Done","0.50"])
ptable.add_row(["~ 100K","Linear SVM","TFIDF Weighted W2V","Done","0.48"])
ptable.add_row(["~ 100K","XGBoost","TFIDF Weighted W2V","NA","0.35"])
ptable.add_row(["~ 100K","XGBoost","TFIDF Weighted W2V","Done","0.34"])
ptable.add_row(["\n","\n","\n","\n","\n"])
ptable.add_row(["~ 400K","Random","TFIDF","NA","0.74"])
ptable.add_row(["~ 400K","Logistic Regression","TFIDF","Done","0.45"])
ptable.add_row(["~ 400K","Linear SVM","TFIDF","Done","0.45"])
print(ptable)
```

Model Comparision				
Dataset Size	Model Name	Tokenizer	Hyperparameter Tunning	Test Log Loss
~ 100K	Random	TFIDF Weighted W2V	NA	0.89
~ 100K	Logistic Regression	TFIDF Weighted W2V	Done	0.50
~ 100K	Linear SVM	TFIDF Weighted W2V	Done	0.48
~ 100K	XGBoost	TFIDF Weighted W2V	NA	0.35
~ 100K	XGBoost	TFIDF Weighted W2V	Done	0.34
~ 400K	Random	TFIDF	NA	0.74
~ 400K	Logistic Regression	TFIDF	Done	0.45
~ 400K	Linear SVM	TFIDF	Done	0.45

As dimension increases Logistic Regression and Linear SVM, starts to perform well,whereas XGBoost produces almost same results after hyperparameter tunning(This can be improved by tunnig more hyperparameters)

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 0.89, that means, the other models has to produce less than 0.89.
2. After that we have applied Logistic Regression on ~100K dataset with hyperparameter tuning, which producs the log loss of 0.50, 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 0.48, which is slightly lower 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.
5. Finally, we applied XGBoost Model on ~100k dataset with hyperparameter tuning, which produces the log loss of 0.34, which is slightly lower than XGBoost Model with no hyperparameter tuning.

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 794.Whereas 'Logistic Regression' and 'Linear SVM' performs moderately on low dimension data.

To test this, we will perform 'Logistic Regression' and 'Linear SVM' on complete ~400 dataset, and we should get better results as compared to above models.

Tokenizer: TFIDF

1. First we have applied simple Random Model(Dumb Model), which gives the log loss of 0.74, that means, the other models has to produce less than 0.74.
2. After that we have applied Logistic Regression on ~400K dataset with hyperparameter tuning, which produces the log loss of 0.45, which is significantly lower than Random Model, also it is lower than previous logistic regression model(performed using TFIDF Weighted W2V).
3. We have applied Linear SVM on ~400K dataset with hyperparameter tuning, which produces the log loss of 0.45, which is similar to Logistic Regression, but it is lower than previous Linear SVM model(performed using TFIDF Weighted W2V).

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'