Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

Problem Statement

- Identify which questions asked on Quora are duplicates of questions that have already been asked.

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

1.2 Sources/Useful Links

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

Useful Links

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning 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 m
arket in india?","What is the step by step guide to invest in sh
are market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamon
d?","What would happen if the Indian government stole the Kohino
or (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?","Ho
w can I see all my Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

In [1]:
```
# import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline

import os
import gc

import re
```

```
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import warnings
warnings.filterwarnings("ignore")
import plotly.offline as py
#py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
# This package is used for finding longest common subsequence between t
wo strings
# you can write your own dp code for this
import distance

from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud
-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
from nltk.corpus import stopwords
import time
from sklearn.preprocessing import normalize
from tqdm import tqdm
import spacy
import pickle
```

```
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os

import datetime as dt

from sklearn.manifold import TSNE


from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
#from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score


from sklearn.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

import xgboost as xgb
from xgboost import XGBClassifier

pd.set_option('display.max_colwidth', -1)
```

3.1 Reading data and basic stats

In [2]:
```
# import data
df = pd.read_csv("train.csv")

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

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

In [4]:
```
pd.set_option('display.max_colwidth', -1)
```

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 share market in india?	What is the step by step guide to invest in share market?	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24} [/math] is divided by 24,23?	0
4	4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
```

```
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404289 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

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

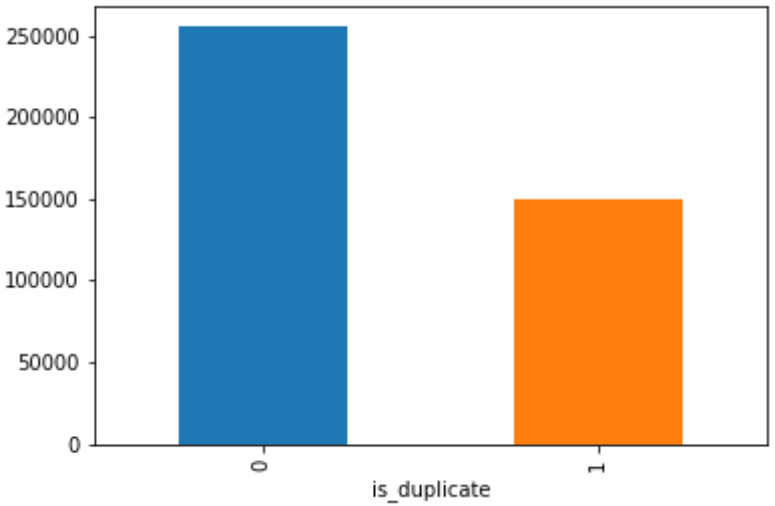
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

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

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ade596a5f8>
```



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

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

In [8]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.f
ormat(100 - round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.for
mat(round(df['is_duplicate'].mean()*100, 2)))
```

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

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

3.2.2 Number of unique questions

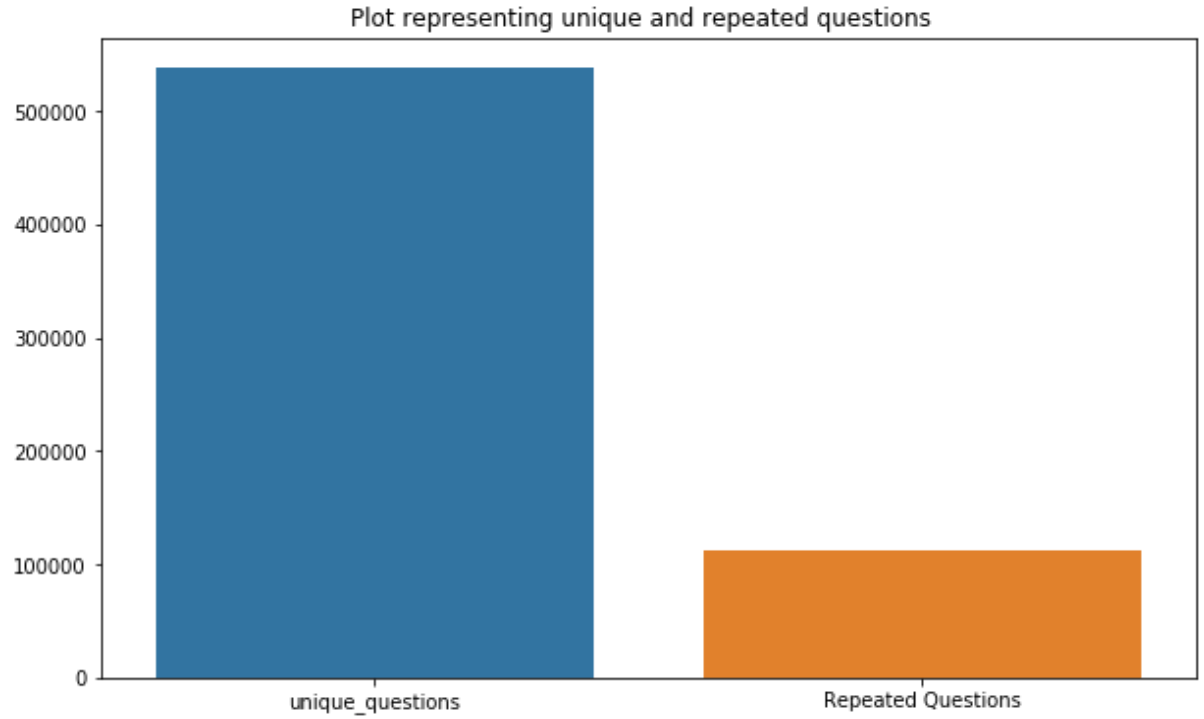
In [9]:
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
```

```
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {} (
{}%)\n'.format(qs_morethan_onetime,qs_morethan_onetime/unique_qs*100))

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.7
7953945937505%)

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

In [10]:

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

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



3.2.3 Checking for Duplicates

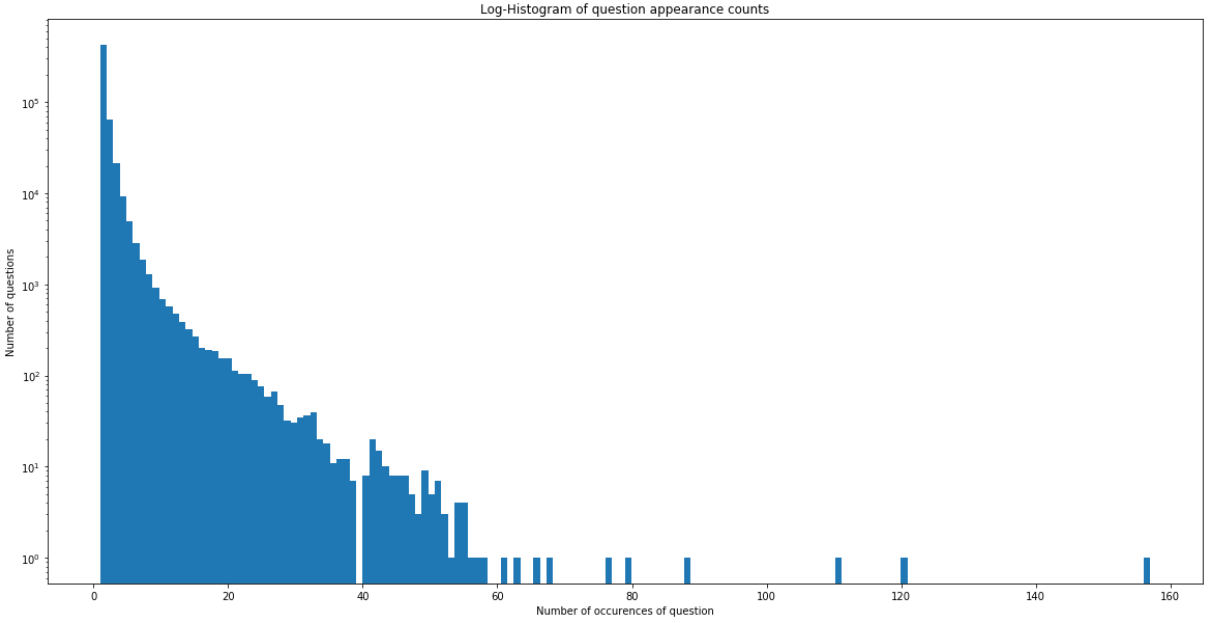
In [11]:
```
#checking whether there are any repeated pair of questions

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
```

3.2.4 Number of occurrences of each question

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

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

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

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

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

plt.ylabel('Number of questions')

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

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



3.2.5 Checking for NULL values

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

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


                                                  question2  \
105780  NaN

201841  NaN

363362  My Chinese name is Haichao Yu. What English name is most suitab
le for me considering the pronounciation of my Chinese name?

        is_duplicate
105780  0
201841  0
363362  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_r
0	0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0	4	1	51	88	

2	2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0	1	1	73	59	

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_r
3	3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24} [/math] is divided by 24,23?	0	1	1	50	65	
4	4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0	3	1	76	39	

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

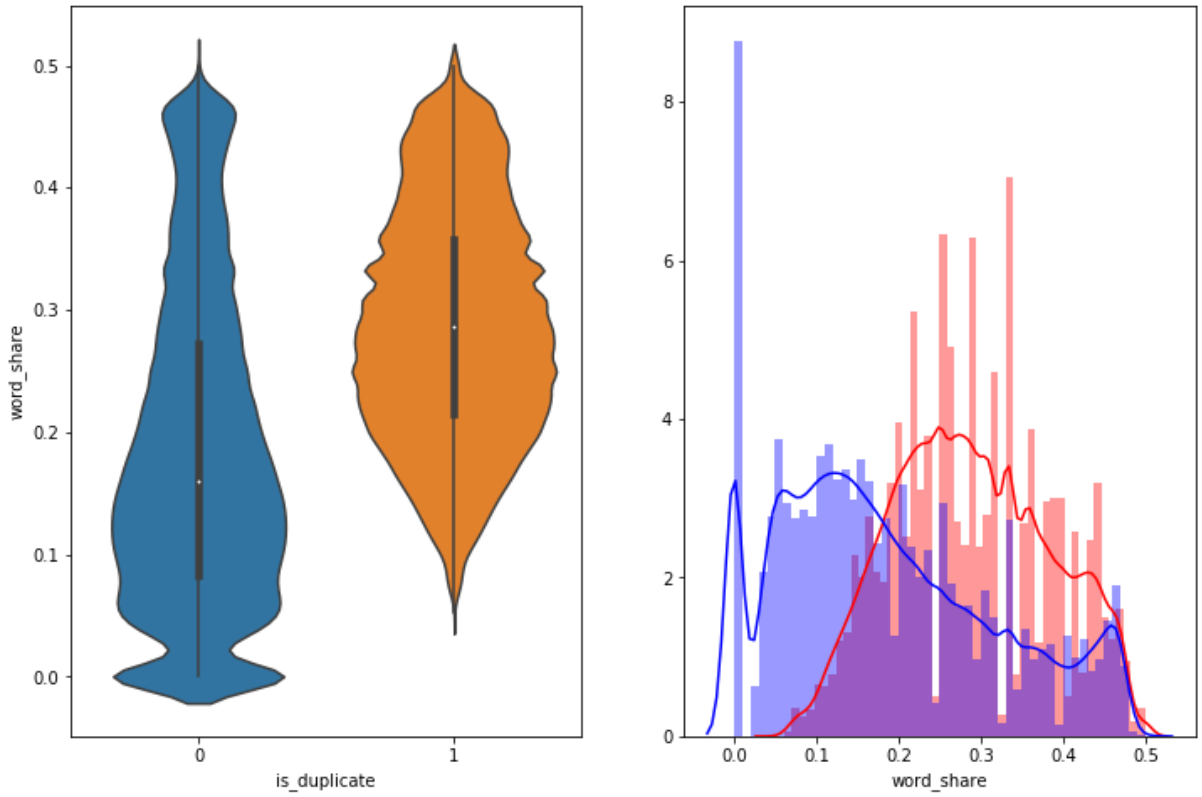
In [16]:
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n
_words']))

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()
```

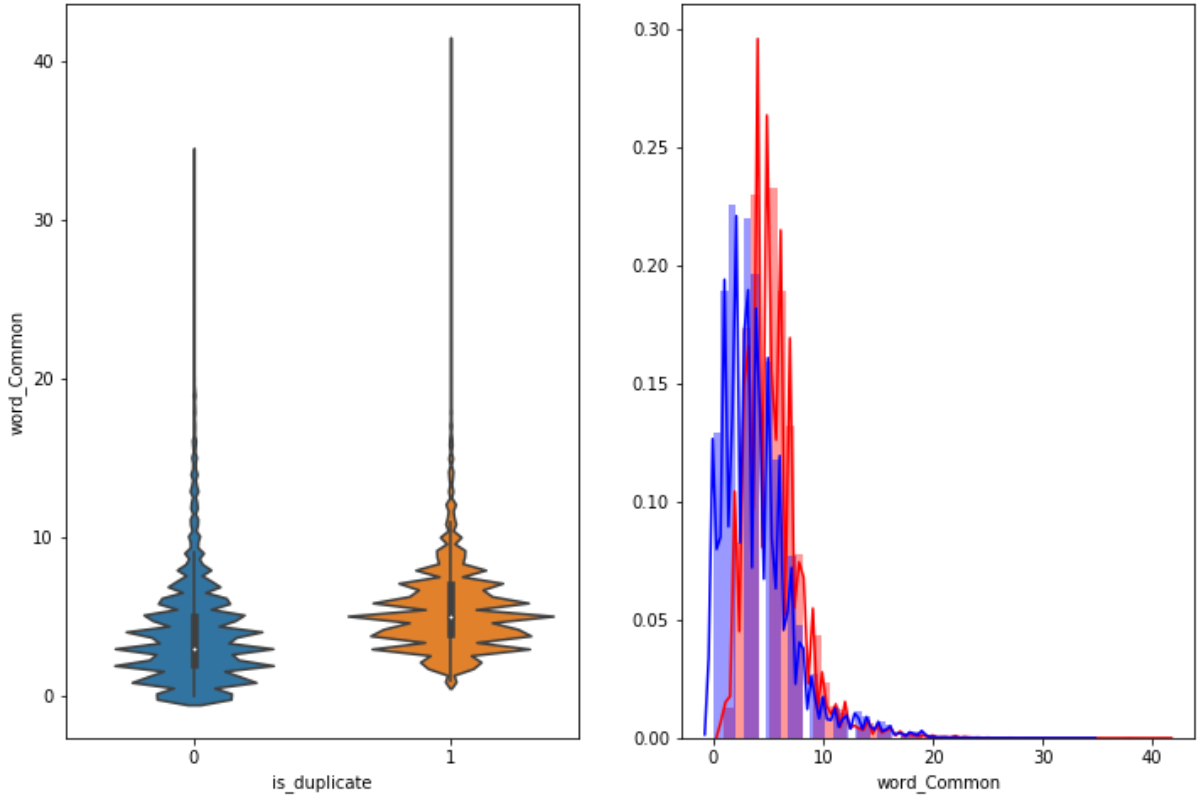


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [18]:
```
### 1.2.1 : EDA: Advanced Feature Extraction.
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

1.2.1 : 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('')
    df.head()
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_v
0	0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0	4	1	51	88	

3.4 Preprocessing of Text

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

3.4 Preprocessing of Text

In [21]:
```
# To get the results in 4 decimal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):

    # convert the text to lowercase
    x = str(x).lower()

    # expanding contractions
    x = x.replace(",000,000", "m").replace(",000", "k").replace("´",
"'").replace("’", "'")\
                           .replace("won't", "will not").replace("canno
t", "can not").replace("can't", "can not")\
                           .replace("n't", " not").replace("what's", "w
hat is").replace("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i a
m").replace("'re", " are")\
                           .replace("he's", "he is").replace("she's",
"she is").replace("'s", " own")\
                           .replace("%", " percent ").replace("₹", " ru
pee ").replace("$", " dollar ")\
                           .replace("€", " euro ").replace("'ll", " wil
l")
```

```
    # converting to million and thousands
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)

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


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


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


    return x
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [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_WORD
S])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORD
S])

    #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(q
2_tokens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q
2_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["question
1"], 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-fu
zz-function-to-compare-2-strings
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_rat
io(x["question1"], x["question2"]), axis=1)
```

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

In [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	.
0	0	1	2	what is the step by step guide to invest in share market in india	what is the step by step guide to invest in share market	0	0.999980	0.833319	0.999983	0.999983	.

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	.
1	1	3	4	what is the story of kohinoor koh i noor diamond	what would happen if the indian government stole the kohinoor koh i noor diamond back	0	0.799984	0.399996	0.749981	0.599988	.

2 rows × 21 columns

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

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

In [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"]]).f
latten()
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["questio
n2"]]).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(te
xtn_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), stopwo
rds=stopwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
```

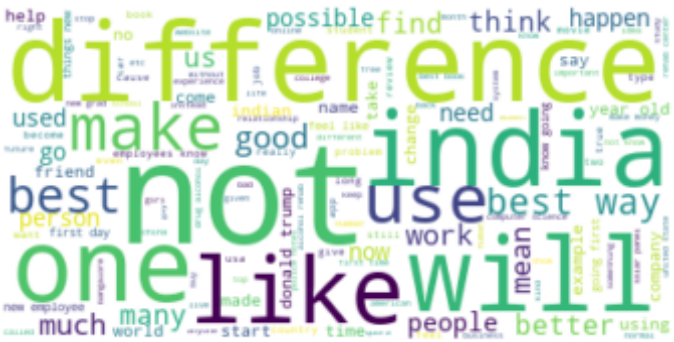
```
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word Cloud for Duplicate Question pairs

Word Clouds generated from non duplicate pair question's text

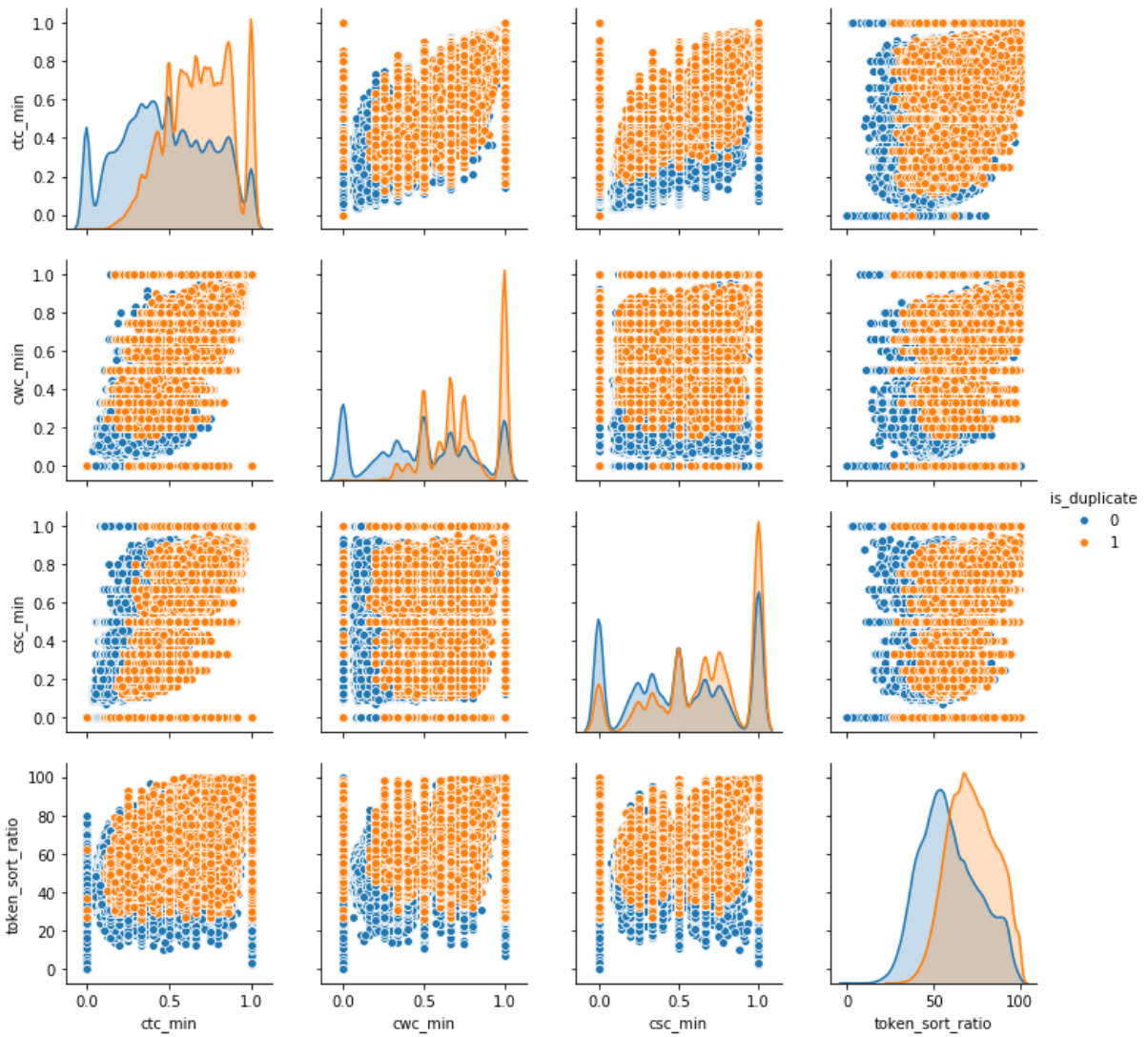
In [27]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwor
ds=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word Cloud for non-Duplicate Question pairs:



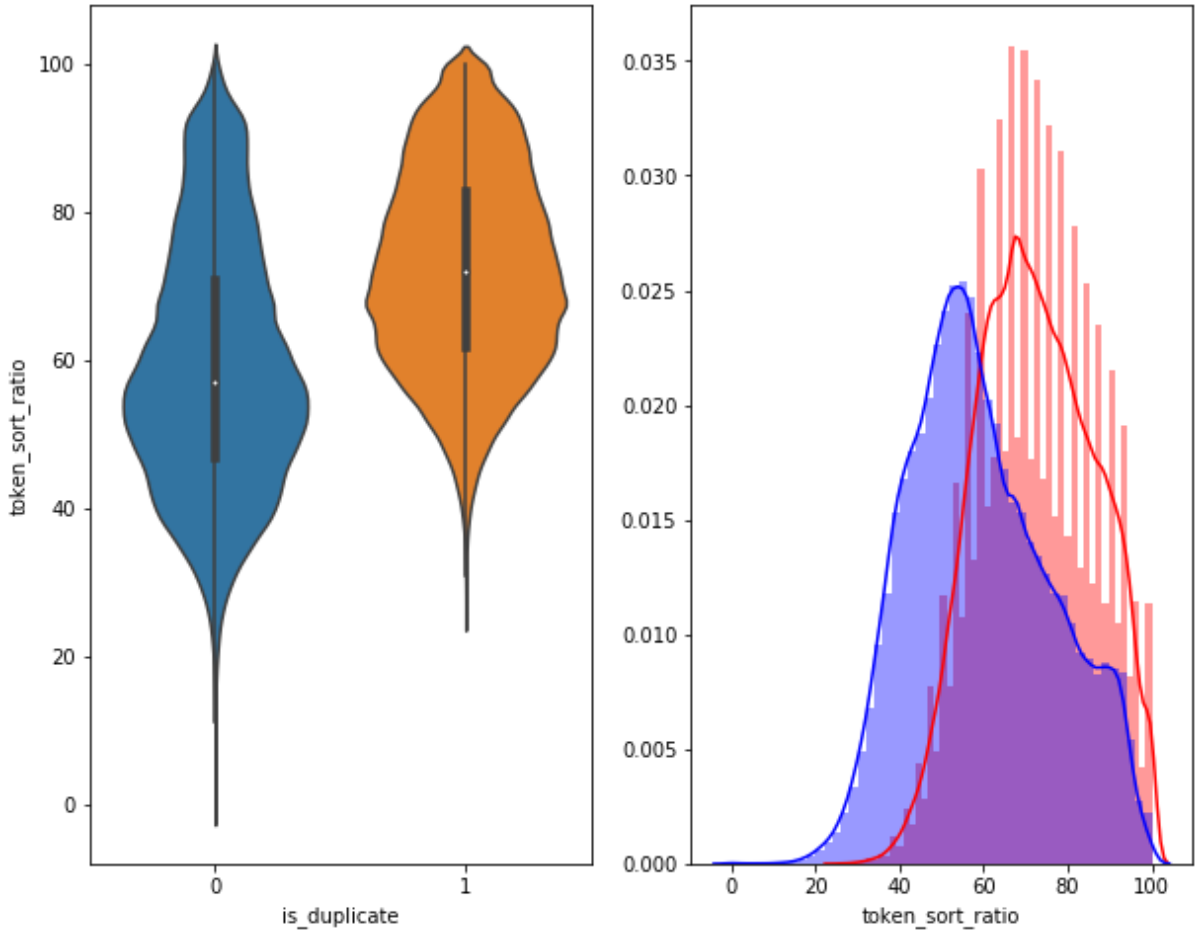
3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

In [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()
```



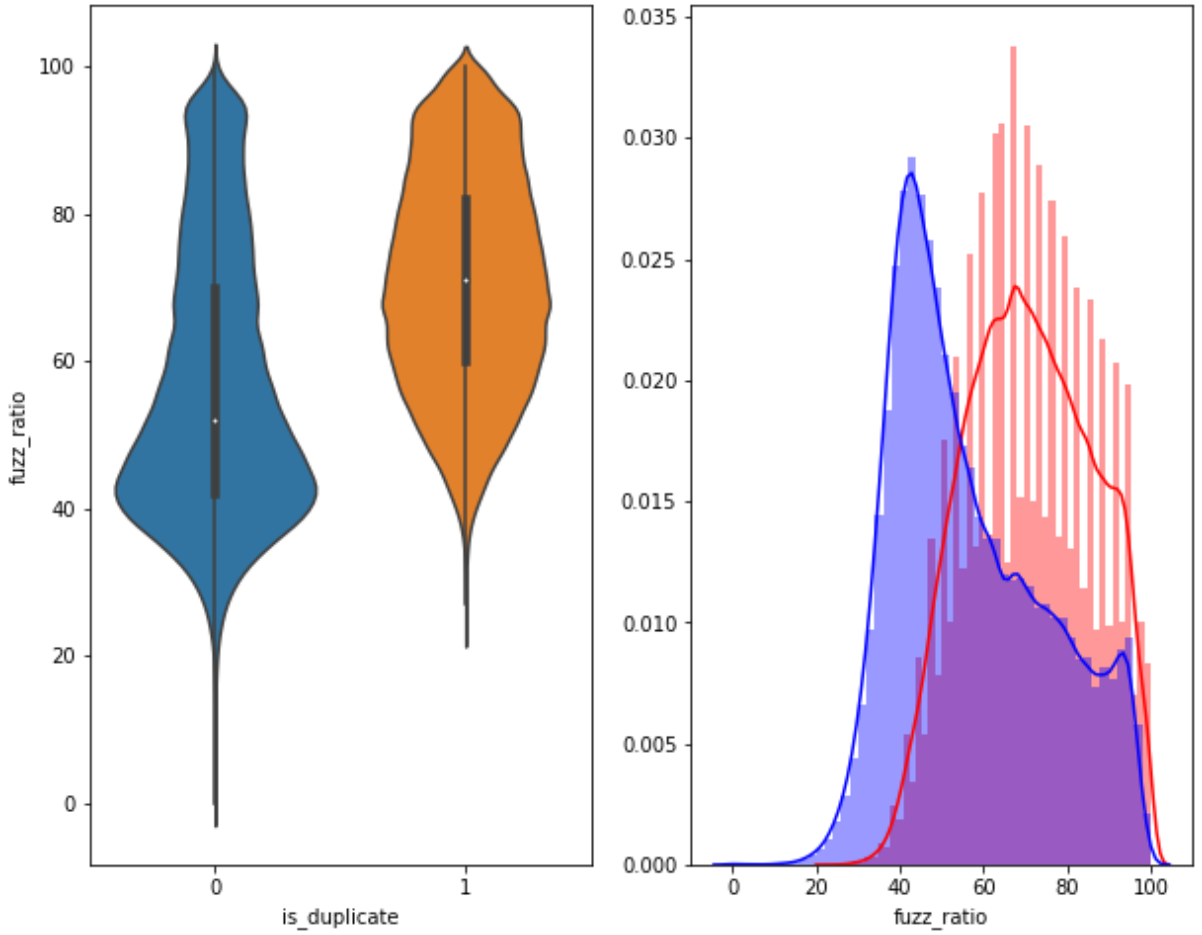
In [29]:
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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:] , la
bel = "1", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , la
bel = "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()
```



3.5.2 Visualization

In [31]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated aft
er cleaning the data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max',
'csc_min', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_w
ord_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)
```

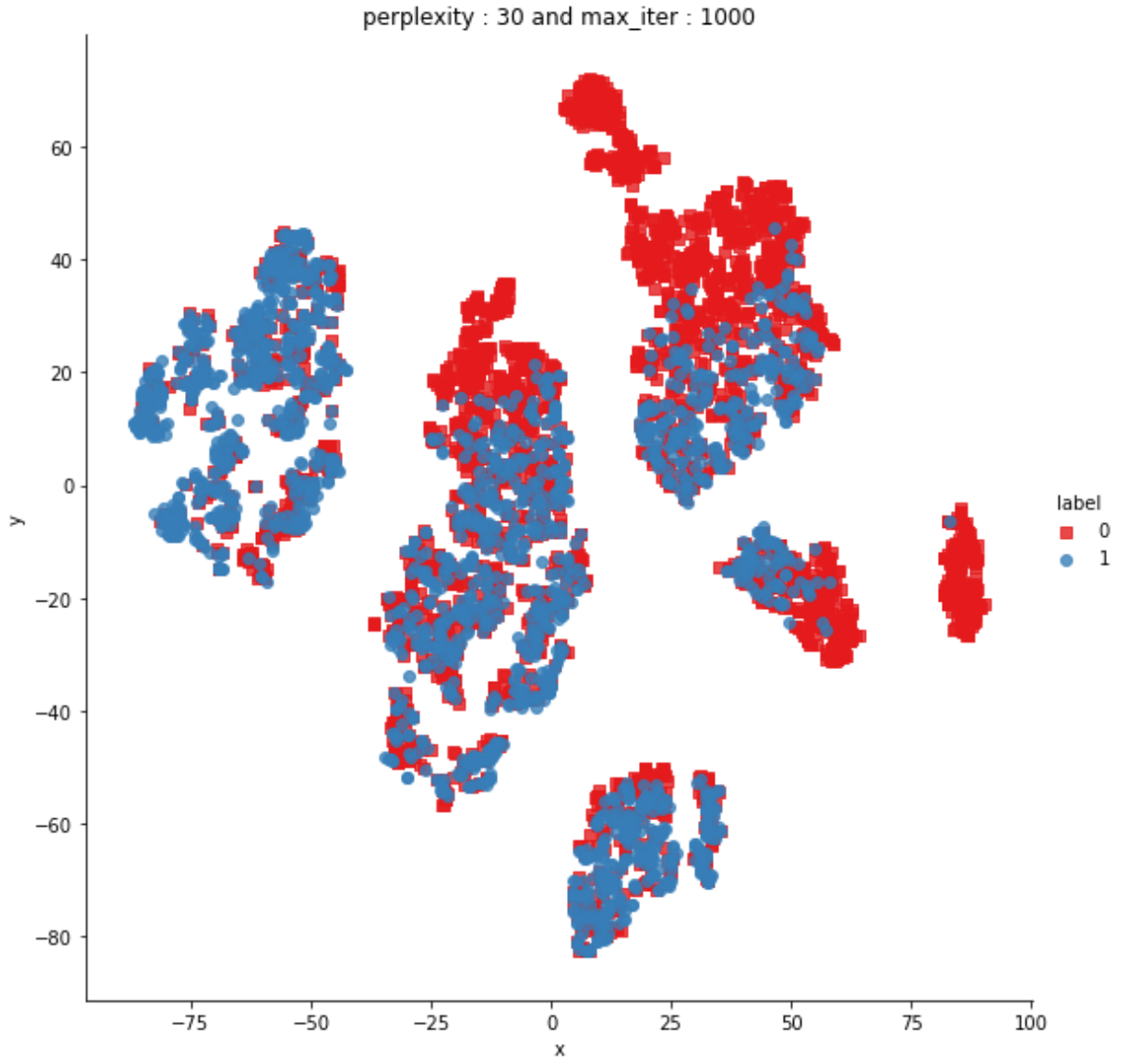
```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.130s...
[t-SNE] Computed neighbors for 5000 samples in 0.333s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional 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 0.243s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50
iterations in 2.296s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (5
```

```
0 iterations in 1.668s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (5
0 iterations in 1.681s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (5
0 iterations in 1.656s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (5
0 iterations in 1.723s)
[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 1.693s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50
iterations in 1.676s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50
iterations in 1.765s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50
iterations in 1.684s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50
iterations in 1.679s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50
iterations in 1.665s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50
iterations in 1.675s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50
iterations in 1.802s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50
iterations in 1.930s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50
iterations in 1.886s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50
iterations in 1.883s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50
iterations in 1.737s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50
iterations in 1.711s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50
iterations in 1.687s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (5
```

```
0 iterations in 1.736s)
[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,pa
lette="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



In [34]:
```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
```

```
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.013s...
[t-SNE] Computed neighbors for 5000 samples in 0.320s...
[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 0.196s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50
iterations in 8.687s)
[t-SNE] Iteration 100: error = 69.1100388, gradient norm = 0.0034323 (5
0 iterations in 4.592s)
[t-SNE] Iteration 150: error = 67.6163483, gradient norm = 0.0017810 (5
0 iterations in 4.361s)
[t-SNE] Iteration 200: error = 67.0578613, gradient norm = 0.0011246 (5
0 iterations in 3.934s)
[t-SNE] Iteration 250: error = 66.7297821, gradient norm = 0.0009272 (5
0 iterations in 4.017s)
[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 5.028s)
[t-SNE] Iteration 350: error = 1.1559117, gradient norm = 0.0001985 (50
iterations in 6.203s)
[t-SNE] Iteration 400: error = 1.0108488, gradient norm = 0.0000976 (50
iterations in 6.385s)
[t-SNE] Iteration 450: error = 0.9391674, gradient norm = 0.0000627 (50
iterations in 6.408s)
[t-SNE] Iteration 500: error = 0.9015961, gradient norm = 0.0000508 (50
iterations in 6.558s)
[t-SNE] Iteration 550: error = 0.8815936, gradient norm = 0.0000433 (50
```

```
iterations in 6.253s)
[t-SNE] Iteration 600: error = 0.8682337, gradient norm = 0.0000373 (50
iterations in 6.317s)
[t-SNE] Iteration 650: error = 0.8589998, gradient norm = 0.0000360 (50
iterations in 6.572s)
[t-SNE] Iteration 700: error = 0.8518325, gradient norm = 0.0000281 (50
iterations in 6.564s)
[t-SNE] Iteration 750: error = 0.8455728, gradient norm = 0.0000284 (50
iterations in 6.284s)
[t-SNE] Iteration 800: error = 0.8401663, gradient norm = 0.0000264 (50
iterations in 6.489s)
[t-SNE] Iteration 850: error = 0.8351609, gradient norm = 0.0000265 (50
iterations in 6.200s)
[t-SNE] Iteration 900: error = 0.8312420, gradient norm = 0.0000225 (50
iterations in 7.494s)
[t-SNE] Iteration 950: error = 0.8273517, gradient norm = 0.0000231 (50
iterations in 6.146s)
[t-SNE] Iteration 1000: error = 0.8240154, gradient norm = 0.0000213 (5
0 iterations in 6.380s)
[t-SNE] KL divergence after 1000 iterations: 0.824015
```

In [35]:

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

data=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered
```

```
 features')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

3.6 Featurizing text data with tfidf weighted word-vectors

In [2]:
```
# avoid decoding problems
df = pd.read_csv("train.csv")

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

In [3]:
```
df.head()
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24} [/math] is divided by 24,23?	0

	id	qid1	qid2	question1	question2	is_duplicate
4	4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0

We will load the features that we have calculated with the train.csv and then merge them. After we merge these features, we will split the data in train and test to avoid data leakage. After this step we will perform vectorization.

- **train.csv** contains the questions q1 and q2, their ids and the corresponding labels from quora
- **df_fe_without_preprocessing_train.csv** contains the above features as well as simple features that are extracted from the questions such as the length of the question, no. of common words etc.
- **nlp_features_train.csv** along with the above features in train.csv, this file contains advanced feature such as token sort ratio, fuzz ration etc

Note The following cells have been executed during previous runs and to avoid memory clogging, I am not running following cells.

In [2]:
```
#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 n
otebook")

#prepro_features_train.csv (Simple Preprocessing Feartures)
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encodi
ng='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or
 run previous notebook")
```

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

Out[4]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	.
0	0	1	2	what is the step by step guide to invest in share market in india	what is the step by step guide to invest in share market	0	0.999980	0.833319	0.999983	0.999983	.
1	1	3	4	what is the story of kohinoor koh i noor diamond	what would happen if the indian government stole the kohinoor koh i noor diamond back	0	0.799984	0.399996	0.749981	0.599988	.
2	2	5	6	how can i increase the speed of my internet connection while using a vpn	how can internet speed be increased by hacking through dns	0	0.399992	0.333328	0.399992	0.249997	.
3	3	7	8	why am i mentally very lonely how can i solve it	find the remainder when math 23 24 math is divided by 24 23	0	0.000000	0.000000	0.000000	0.000000	.

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	.
4	4	9	10	which one dissolve in water quikly sugar salt methane and carbon di oxide	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.666644	

5 rows × 21 columns

In [20]:
```
dfppro.head()
```

Out[20]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_r
0	0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0	4	1	51	88	

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_r
2	2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0	1	1	73	59	
3	3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24} [/math] is divided by 24,23?	0	1	1	50	65	
4	4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0	3	1	76	39	

In [18]:
```
dfppro.shape
```

Out[18]: (404290, 17)

We merge both these features based on id

In [3]:
```
df_merged = pd.merge(dfnlp, dfppro, on = "id")
```

In [8]:
```
df_merged.shape
```

Out[8]: (404290, 37)

In [9]:
```
df_merged.head()
```

Out[9]:

	id	qid1_x	qid2_x	question1_x	question2_x	is_duplicate_x	cwc_min	cwc_max	csc_min	c
0	0	1	2	what is the step by step guide to invest in share market in india	what is the step by step guide to invest in share market	0	0.999980	0.833319	0.999983	0
1	1	3	4	what is the story of kohinoor koh i noor diamond	what would happen if the indian government stole the kohinoor koh i noor diamond back	0	0.799984	0.399996	0.749981	0
2	2	5	6	how can i increase the speed of my internet connection while using a vpn	how can internet speed be increased by hacking through dns	0	0.399992	0.333328	0.399992	0
3	3	7	8	why am i mentally very lonely how can i solve it	find the remainder when math 23 24 math is divided by 24 23	0	0.000000	0.000000	0.000000	0
4	4	9	10	which one dissolve in water quikly sugar salt methane and carbon di oxide	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0

5 rows × 37 columns

In [4]:
```
# We assign features to X and the response to y variables respectively
y = df_merged["is_duplicate_x"]
X = df_merged.drop("is_duplicate_x", axis = 1)
```

In [11]:
```
X.shape
```

Out[11]:
```
(404290, 36)
```

In [16]:
```
X.columns
```

Out[16]:
```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', 'cwc_mi
n',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_wor
d_eq',
       'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio',
       'token_sort_ratio', 'fuzz_ratio', 'fuzz_partial_ratio',
       'longest_substr_ratio', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', 'freq_qid1', 'freq_qid2', 'q1le
n',
       'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Tota
l',
       'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

In [12]:
```
len(y)
```

Out[12]:
```
404290
```

In [7]:
```
# 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(X.columns)
#for i in cols:
 #   X[i] = X[i].apply(pd.to_numeric)
 #   print(i)
```

Splitting the data before we perform vectorization of questions to avoid data leakage

In [5]:
```
X_train,X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.3)
```

In [6]:
```
print("Shape of Train data: ", X_train.shape)
print("Shape of Test data: ", X_test.shape)
```

```
Shape of Train data:  (283003, 36)
Shape of Test data:  (121287, 36)
```

In [9]:
```
X_train.columns
```

Out[9]:
```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', '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', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', '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')
```

Vectorization

A. TFIDF-weighted word2vec vectorization

Training set w2v for question1

In [7]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
```

```
nlp = spacy.load('en_core_web_sm')
```

In [ ]:
```
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_train['question1_x'].astype("unicode"))):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    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)
```

In [83]:
```
#filename='vecs1_train_q1.sav'
#pickle.dump(vecs1,open(filename,'wb'))
```

In [8]:
```
vecs1=pickle.load(open('vecs1_train_q1.sav','rb'))
```

In [9]:
```
X_train['q1_feats_m_train'] = list(vecs1)
```

In [18]:
```
X_train.shape
```

Out[18]:
```
(283003, 38)
```

In [19]:
```
X_train.columns
```

Out[19]:
```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', 'cwc_mi
n',
```

```
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_wor
d_eq',
       'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio',
       'token_sort_ratio', 'fuzz_ratio', 'fuzz_partial_ratio',
       'longest_substr_ratio', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', 'freq_qid1', 'freq_qid2', 'q1le
n',
       'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Tota
l',
       'word_share', 'freq_q1+q2', 'freq_q1-q2', 'q1_feats_m_train',
       'q2_feats_m_train'],
      dtype='object')
```

Training set w2v for question2

In [14]:
```
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_train['question2_x'].astype("unicode"))):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    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)
```

```
100%|████████████████████████████████████████████████████████████████
████████| 283003/283003 [32:48<00:00, 143.76it/s]
```

In [15]:
```
#filename='vecs2_train_q2.sav'
#pickle.dump(vecs1,open(filename,'wb'))
```

In [10]:
```
vecs1=pickle.load(open('vecs2_train_q2.sav','rb'))
```

In [11]:
```
X_train['q2_feats_m_train'] = list(vecs1)
```

Testing set w2v for question1

In [17]:
```
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_test['question1_x'].astype("unicode"))):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    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)
```

```
100%|██████████| 121287/121287 [14:25<00:00, 140.06it/s]
```

In [18]:
```
#filename='vecs1_test_q1.sav'
#pickle.dump(vecs1,open(filename,'wb'))
```

In [12]:
```
del vecs1
```

In [13]:
```
vecs1=pickle.load(open('vecs1_test_q1.sav','rb'))
```

In [14]:
```
X_test['q1_feats_m_test'] = list(vecs1)
```

Testing set w2v for question2

In [20]:
```
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_test['question2_x'].astype("unicode"))):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    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)
```

```
100%|████████████████████████████████████████████████████████████████| 121287/121287 [14:20<00:00, 140.97it/s]
```

In [21]:
```
#filename='vecs2_test_q2.sav'
#pickle.dump(vecs1,open(filename,'wb'))
```

In [15]:
```
del vecs1
```

In [16]:
```
vecs1=pickle.load(open('vecs2_test_q2.sav','rb'))
```

In [17]:
```
X_test['q2_feats_m_test'] = list(vecs1)
```

We will drop certain columns which are related to questions and are in the text format before we do modelling

In [18]:
```
# training data
X_train_ww2v =  X_train.drop(["id", "qid1_x", "qid2_x", "question1_x",
"question2_x","qid1_y", "qid2_y", "question1_y", "question2_y", "is_dup
licate_y"], axis = 1)
X_train_ww2v.shape
```

Out[18]:
```
(283003, 28)
```

In [19]:
```
# test data
X_test_ww2v =  X_test.drop(["id", "qid1_x", "qid2_x", "question1_x", "q
uestion2_x","qid1_y", "qid2_y", "question1_y", "question2_y", "is_dupli
cate_y"], axis = 1)
X_test_ww2v.shape
```

Out[19]:
```
(121287, 28)
```

In [58]:
```
cols = list(X_train_ww2v.columns)
for i in tqdm(cols):
    X_train_ww2v[i] = X_train_ww2v[i].apply(pd.to_numeric)
    #print(i)
```

```
100%|████████████████████████████████████████████████████████████
████████████| 28/28 [00:19<00:00,  1.17it/s]
```

In [59]:
```
cols = list(X_test_ww2v.columns)
for i in tqdm(cols):
    X_test_ww2v[i] = X_test_ww2v[i].apply(pd.to_numeric)
    #print(i)
```

```
100%|████████████████████████████████████████████████████████████
████████████| 28/28 [00:08<00:00,  2.71it/s]
```

In [31]:
```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
y_train = list(map(int, y_train.values))
```

In [60]:
```
# filename='y_train.sav'
# pickle.dump(y_train, open(filename,'wb'))
```

In [7]:
```
# filename='y_test.sav'
# pickle.dump(y_test, open(filename,'wb'))
```

In [88]:
```
y_train = pickle.load(open('y_train.sav','rb'))
y_test = pickle.load(open('y_test.sav','rb'))
```

In [32]:
```
# filename='X_train_ww2v_vectorized.sav'
# pickle.dump(X_train_ww2v,open(filename,'wb'))
```

In [33]:
```
# filename='X_test_ww2v_vectorized.sav'
# pickle.dump(X_test_ww2v,open(filename,'wb'))
```

In [96]:
```
X_train_ww2v = pickle.load(open('X_train_ww2v_vectorized.sav','rb'))
X_test_ww2v = pickle.load(open('X_test_ww2v_vectorized.sav','rb'))
```

In [11]:
```
type(X_train_ww2v)
```

Out[11]:
```
pandas.core.frame.DataFrame
```

In [97]:
```
print("Number of data points in train data :",X_train_ww2v.shape)
print("Number of data points in test data :",X_test_ww2v.shape)
```

```
Number of data points in train data : (283003, 28)
Number of data points in test data : (121287, 28)
```

In [20]:
```
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_dis
tr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in train data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

In [2]:

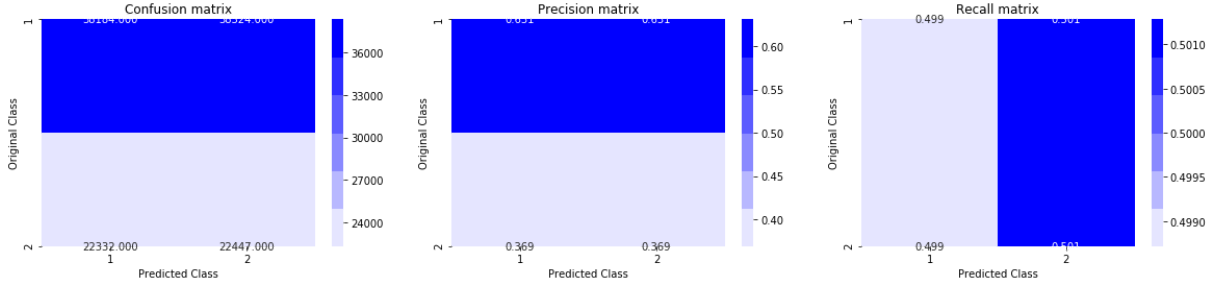
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of element
s 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 correspo
nds 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 element
s in that row
```

```
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
nds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

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

    plt.show()
```

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

In [22]:
```
# 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, predi
cted_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 0.8850088284393789
```



In [26]:
```
type(X_train_ww2v)
```

Out[26]:
```
pandas.core.frame.DataFrame
```

Converting df to array and reshaping data

In [36]:
```
X_train_np = X_train_ww2v.index.get_values()
X_train_np = X_train_np.reshape(-1, 1)
```

In [40]:
```
X_test_np = X_test_ww2v.index.get_values()
```

```
X_test_np = X_test_np.reshape(-1, 1)
```

In [44]:
```
type(X_train_np)
```

Out[44]:
```
numpy.ndarray
```

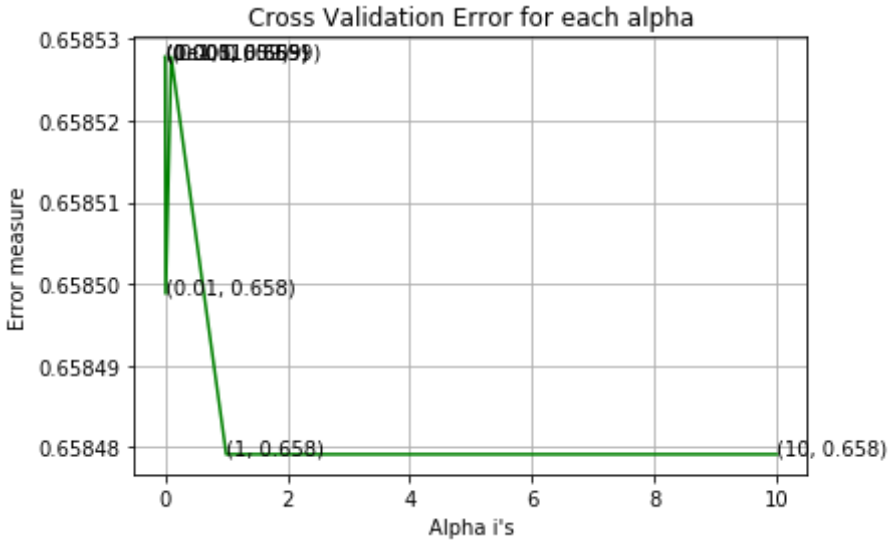
4.4 Logistic Regression with hyperparameter tuning

In [43]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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


log_error_array=[]
for i in tqdm(alpha):
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(X_train_np, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

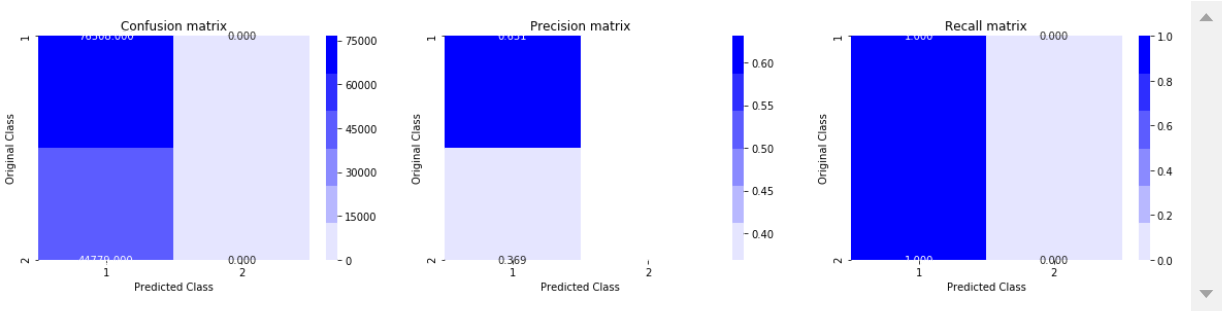
```
    sig_clf.fit(X_train_np, y_train)
    predict_y = sig_clf.predict_proba(X_test_np)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, predict_y, labels=clf.classes_, eps=1e-15))

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_np, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_np, y_train)

predict_y = sig_clf.predict_proba(X_train_np)
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_np)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
  0%|
                                | 0/7 [00:00<?, ?it/s]
```

```
For values of alpha =  1e-05 The log loss is: 0.6585278256347589
 14%|          | 1/7 [01:33<09:19, 93.28s/it]
For values of alpha =  0.0001 The log loss is: 0.6585278256347589
 29%|          | 2/7 [02:57<07:32, 90.55s/it]
For values of alpha =  0.001 The log loss is: 0.6585278256347589
 43%|          | 3/7 [04:33<06:08, 92.13s/it]
For values of alpha =  0.01 The log loss is: 0.6584988168101666
 57%|          | 4/7 [05:59<04:31, 90.50s/it]
For values of alpha =  0.1 The log loss is: 0.6585278256347589
 71%|          | 5/7 [07:21<02:55, 87.81s/it]
For values of alpha =  1 The log loss is: 0.6584791279823448
 86%|          | 6/7 [08:39<01:24, 84.72s/it]
For values of alpha =  10 The log loss is: 0.6584791279811694
100%|          | 7/7 [10:04<00:00, 84.81s/it]
```




```
For values of best alpha =  10 The train log loss is: 0.658493177735093
6
For values of best alpha =  10 The test log loss is: 0.6584791279811694
Total number of data points : 121287
```



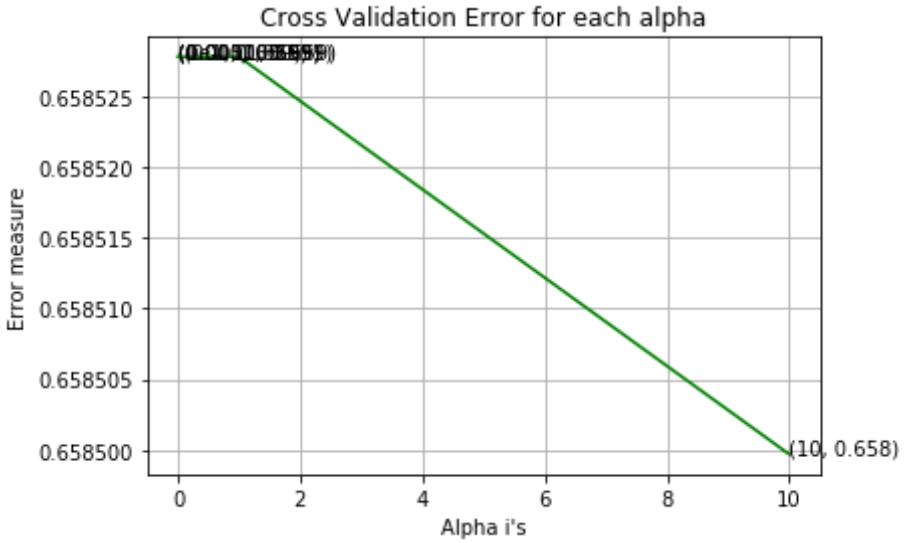
4.5 Linear SVM with hyperparameter tuning

In [45]:

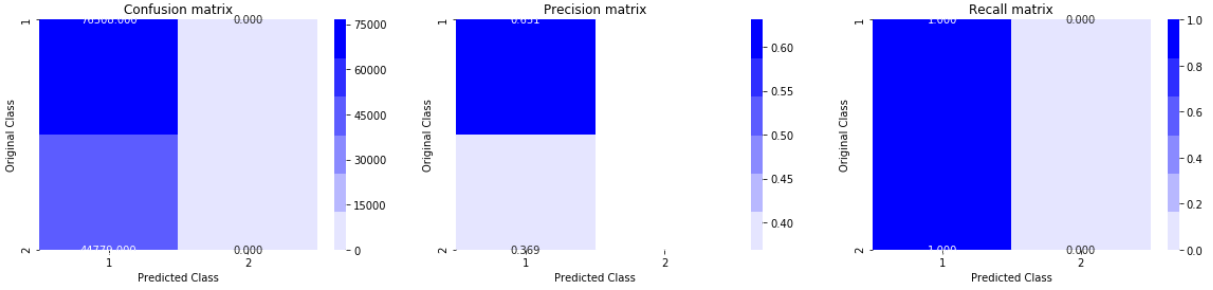
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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_sta
te=42)
    clf.fit(X_train_np, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_np, y_train)
    predict_y = sig_clf.predict_proba(X_test_np)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, 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_np, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_np, y_train)

predict_y = sig_clf.predict_proba(X_train_np)
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_np)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss 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: 0.6585278256347589
For values of alpha =  0.0001 The log loss is: 0.6585278256322724
For values of alpha =  0.001 The log loss is: 0.6585278256294329
For values of alpha =  0.01 The log loss is: 0.6585278256294328
For values of alpha =  0.1 The log loss is: 0.6585278256322725
For values of alpha =  1 The log loss is: 0.6585278256294309
For values of alpha =  10 The log loss is: 0.6584996928565379
```




```
For values of best alpha =  10 The train log loss is: 0.658505063668041
For values of best alpha =  10 The test log loss is: 0.6584996928565379
Total number of data points : 121287
```



4.6 XGBoost

In [46]:
```
import xgboost as xgb
```

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

d_train = xgb.DMatrix(X_train_np, label=y_train)
d_test = xgb.DMatrix(X_test_np, label=y_test)

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

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

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

```
[0]     train-logloss:0.691842  valid-logloss:0.691879
Multiple eval metrics have been passed: 'valid-logloss' will be used fo
r early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.681018  valid-logloss:0.680883
[20]    train-logloss:0.673513  valid-logloss:0.673522
[30]    train-logloss:0.668668  valid-logloss:0.668474
[40]    train-logloss:0.665108  valid-logloss:0.665248
[50]    train-logloss:0.662839  valid-logloss:0.663022
[60]    train-logloss:0.661526  valid-logloss:0.661401
[70]    train-logloss:0.660192  valid-logloss:0.660435
[80]    train-logloss:0.65966   valid-logloss:0.659864
[90]    train-logloss:0.659104  valid-logloss:0.659353
[100]   train-logloss:0.658816  valid-logloss:0.659039
[110]   train-logloss:0.658634  valid-logloss:0.658917
[120]   train-logloss:0.6582    valid-logloss:0.658749
[130]   train-logloss:0.65829   valid-logloss:0.658666
[140]   train-logloss:0.658247  valid-logloss:0.658664
[150]   train-logloss:0.657927  valid-logloss:0.658607
[160]   train-logloss:0.658071  valid-logloss:0.658529
```

```
[170]   train-logloss:0.658119  valid-logloss:0.658422
[180]   train-logloss:0.657699  valid-logloss:0.658387
[190]   train-logloss:0.657988  valid-logloss:0.658471
[200]   train-logloss:0.658132  valid-logloss:0.658497
Stopping. Best iteration:
[180]   train-logloss:0.657699  valid-logloss:0.658387

The test log loss is: 0.6584765733207835
```

B. TFIDF vectorization

Training Data

In [64]:
```
# merge texts
questions = np.array(X_train['question1_x'].astype("unicode")) + np.arr
ay(X_train['question2_x'].astype("unicode"))

tfidf = TfidfVectorizer(lowercase=False,max_features=1000)
X_train_tfidf = tfidf.fit_transform(questions.astype("unicode"))

# dict key:word and value:tf-idf score
word2tfidf_train = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [65]:
```
print("Shape of train TFIDF vectorizer: ", X_train_tfidf.shape)
```

```
Shape of train TFIDF vectorizer:  (283003, 1000)
```

In [52]:
```
X_train.columns
```

Out[52]:
```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', 'cwc_mi
n',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_wor
d_eq',
       'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio',
       'token_sort_ratio', 'fuzz_ratio', 'fuzz_partial_ratio',
       'longest_substr_ratio', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', 'freq_qid1', 'freq_qid2', 'q1le
```

```
n',
       'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Tota
l',
       'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

We will drop question related columns as we now have their vectorized form.

In [66]:
```
X_train_after_dropped_features =  X_train.drop(["id", "qid1_x", "qid2_
x", "question1_x", "question2_x","qid1_y", "qid2_y", "question1_y", "qu
estion2_y", "is_duplicate_y"], axis = 1)
X_train_after_dropped_features.shape
```

Out[66]:
```
(283003, 26)
```

In [67]:
```
X_train_after_dropped_features.columns
```

Out[67]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_ma
x',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq
_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

Now that we have vectorized form of questions, we need to use the other features from X_train and stack them vertically with these vectors

In [68]:
```
from scipy.sparse import hstack
X_train_tfidf = hstack([X_train_tfidf, X_train_after_dropped_features],
 format = "csr")
```

In [69]:
```
X_train_tfidf.shape
```

Out[69]:
```
(283003, 1026)
```

Testing data

In [70]:
```
# merge texts
questions = np.array(X_test['question1_x'].astype("unicode")) + np.arra
y(X_test['question2_x'].astype("unicode"))

tfidf = TfidfVectorizer(lowercase=False,max_features=1000)
X_test_tfidf = tfidf.fit_transform(questions.astype("unicode"))

# dict key:word and value:tf-idf score
word2tfidf_test = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [71]:
```
print("Shape of Test TFIDF vectorizer: ", X_test_tfidf.shape)
```

```
Shape of Test TFIDF vectorizer:  (121287, 1000)
```

In [72]:
```
X_test.columns
```

Out[72]:
```
Index(['id', 'qid1_x', 'qid2_x', 'question1_x', 'question2_x', 'cwc_mi
n',
       'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_wor
d_eq',
       'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio',
       'token_sort_ratio', 'fuzz_ratio', 'fuzz_partial_ratio',
       'longest_substr_ratio', 'qid1_y', 'qid2_y', 'question1_y',
       'question2_y', 'is_duplicate_y', 'freq_qid1', 'freq_qid2', 'q1le
n',
       'q2len', 'q1_n_words', 'q2_n_words', 'word_Common', 'word_Tota
l',
       'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

We will drop question related columns as we now have their vectorized form.

In [73]:
```
X_test_after_dropped_features =  X_test.drop(["id", "qid1_x", "qid2_x",
 "question1_x", "question2_x","qid1_y", "qid2_y", "question1_y", "quest
```

```
ion2_y", "is_duplicate_y"], axis = 1)
X_test_after_dropped_features.shape
```

Out[73]:
```
(121287, 26)
```

In [74]:
```
X_test_after_dropped_features.columns
```

Out[74]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_ma
x',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq
_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

Now that we have vectorized form of questions, we need to use the other features from X_test and stack them vertically with these vectors

In [75]:
```
from scipy.sparse import hstack
X_test_tfidf = hstack([X_test_tfidf, X_test_after_dropped_features], fo
rmat = "csr")
```

In [76]:
```
X_test_tfidf.shape
```

Out[76]:
```
(121287, 1026)
```

Load files here for tfidf vectorization

In [77]:
```
#filename='X_train_tfidf.sav'
#pickle.dump(X_train_tfidf,open(filename,'wb'))
```

In [78]:
```
#filename='X_test_tfidf.sav'
#pickle.dump(X_test_tfidf,open(filename,'wb'))
```

In [89]:
```
X_train_tfidf = pickle.load(open('X_train_tfidf.sav','rb'))
X_test_tfidf = pickle.load(open('X_test_tfidf.sav','rb'))
```

In [22]:
```
# 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
# test set

#cols = list(X_train_tfidf.columns)
#for i in tqdm(cols):
 #   X_train_tfidf[i] = X_train_tfidf[i].apply(pd.to_numeric)
    #print(i)
```

In [23]:
```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-int
y_train = list(map(int, y_train.values))
```

- 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.

Modeling

In [57]:
```
print("Number of data points in train data :",X_train_tfidf.shape)
print("Number of data points in test data :",X_test_tfidf.shape)
```

```
Number of data points in train data : (283003, 10026)
Number of data points in test data : (121287, 10026)
```

In [90]:
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_
distr[1])/train_len)
```

```
print("-"*10, "Distribution of output variable in test data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_dis
tr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in test data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

In [10]:

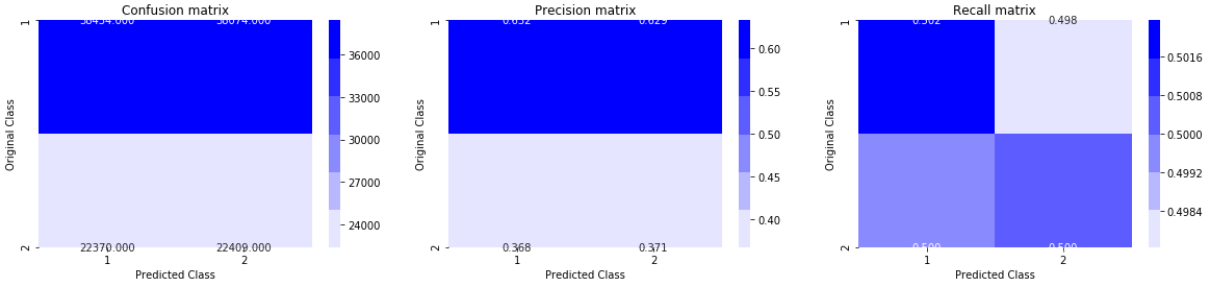
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of element
s 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 correspo
nds 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 element
s in that row
    # C = [[1, 2],
```

```
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
nds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

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

    plt.show()
```

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

In [60]:
```
# 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, predi
cted_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 0.8828023393918247
```



4.4 Logistic Regression

In [79]:
```
type(X_train_tfidf)
```

Out[79]:
```
scipy.sparse.csr.csr_matrix
```

Converting sparse matrix to dense

In [82]:
```
X_train_tfidf = X_train_tfidf.todense()
```

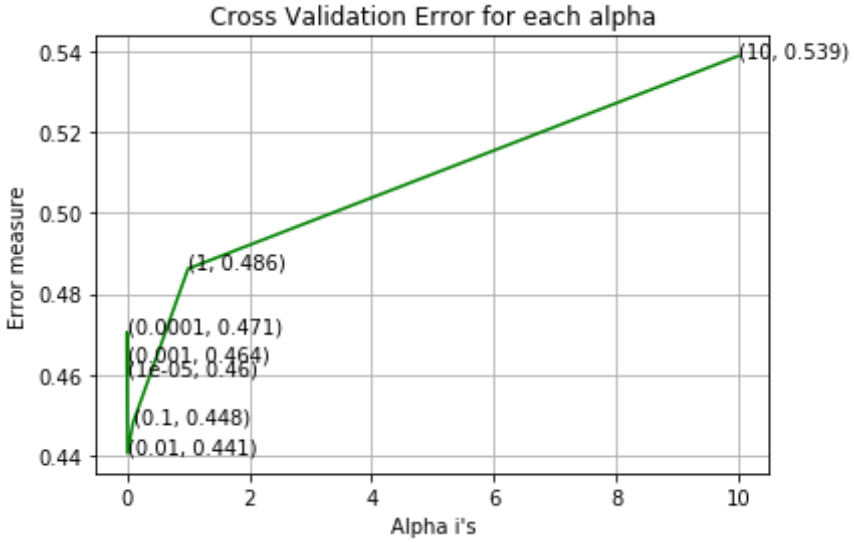
```
X_test_tfidf = X_test_tfidf.todense()
```

In [84]:
```
#X_train_np = X_train_tfidf.index.get_values()
#X_train_np = X_train_np.reshape(-1, 1)
```

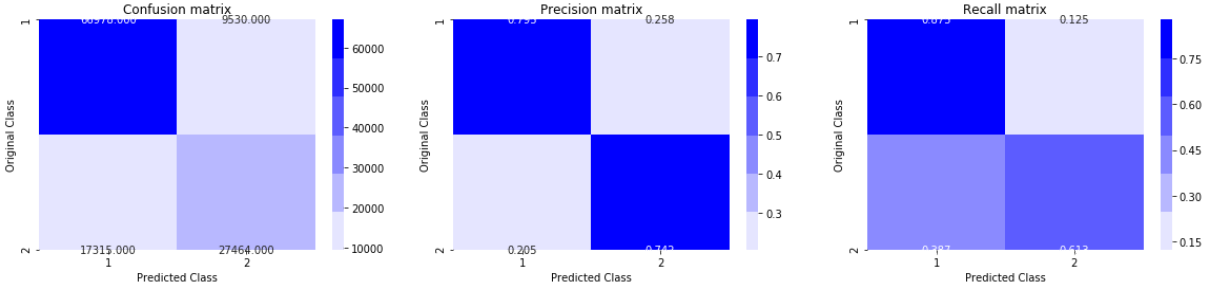
In [85]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.class
es_, eps=1e-15))
```

```
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, 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_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_tfidf, y_train)

predict_y = sig_clf.predict_proba(X_train_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss 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: 0.4601659285802642
For values of alpha =  0.0001 The log loss is: 0.47054018904200623
For values of alpha =  0.001 The log loss is: 0.46375063450187787
For values of alpha =  0.01 The log loss is: 0.4406110789291779
For values of alpha =  0.1 The log loss is: 0.44817377481955234
For values of alpha =  1 The log loss is: 0.48617842035225495
For values of alpha =  10 The log loss is: 0.5387401633996405
```




```
For values of best alpha =  0.01 The train log loss is: 0.4374815364252
173
For values of best alpha =  0.01 The test log loss is: 0.44061107892917
79
Total number of data points : 121287
```

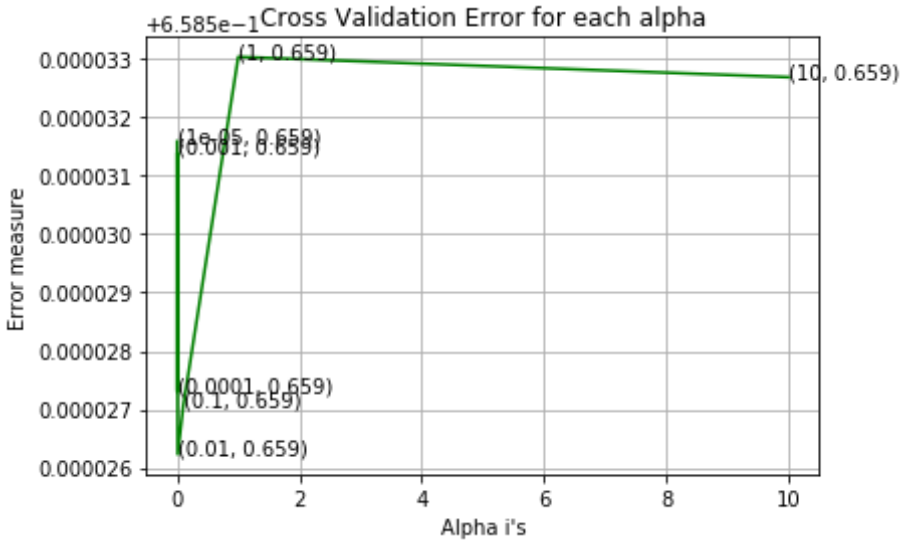


4.5 Linear SVM

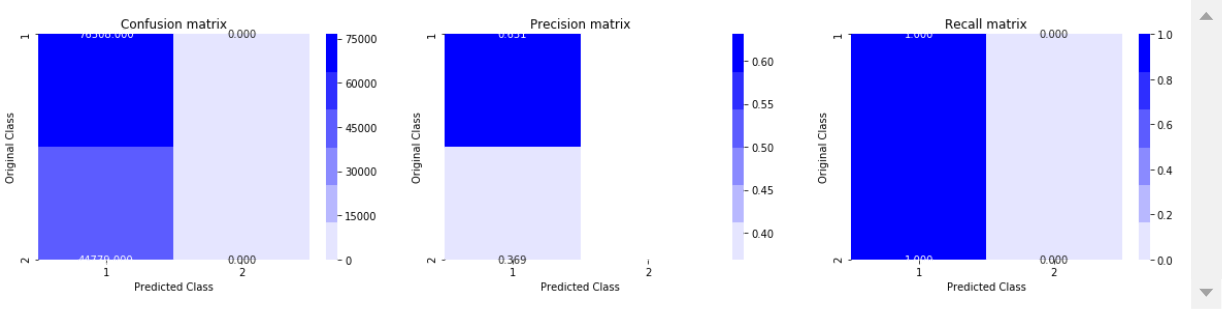
In [93]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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_sta
te=42)
    clf.fit(X_train_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, 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_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_tfidf, y_train)

predict_y = sig_clf.predict_proba(X_train_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss 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: 0.6585315868731638
For values of alpha =  0.0001 The log loss is: 0.6585273127510437
For values of alpha =  0.001 The log loss is: 0.6585313933152336
For values of alpha =  0.01 The log loss is: 0.6585262309584129
For values of alpha =  0.1 The log loss is: 0.658527057311145
For values of alpha =  1 The log loss is: 0.6585330352130937
For values of alpha =  10 The log loss is: 0.6585326868875435
```




```
For values of best alpha =  0.01 The train log loss is: 0.6585274280770
488
For values of best alpha =  0.01 The test log loss is: 0.65852623095841
29
Total number of data points : 121287
```



Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss

In [18]:
```
# Referred
# https://www.kaggle.com/stuarthallows/using-xgboost-with-scikit-learn
# https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9
100a3b2f
# https://www.kaggle.com/nikitpatel/random-grid-bayes-search-cv-for-xgb
#Random-Search-CV-Approach-for-classification
# https://www.datacamp.com/community/tutorials/xgboost-in-python
```

In [3]:
```
import xgboost as xbg
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
```

In [4]:
```
X_train_ww2v = pickle.load(open('X_train_ww2v_vectorized.sav','rb'))
X_test_ww2v = pickle.load(open('X_test_ww2v_vectorized.sav','rb'))
y_train=pickle.load(open('y_train.sav','rb'))
y_test = pickle.load(open('y_test.sav','rb'))
```

In [5]:
```
X_train_np = X_train_ww2v.index.get_values()
X_train_np = X_train_np.reshape(-1, 1)
```

In [6]:
```
X_test_np = X_test_ww2v.index.get_values()
X_test_np = X_test_np.reshape(-1, 1)
```

In [ ]:
```
#params['objective'] = 'binary:logistic'
#params['eval_metric'] = 'logloss'
```

In [49]:
```
# defining parameters

params = {
          "max_depth": range(3, 11),
           "eta ": uniform(0.0001, 0.1)
```

```
        }
```

In [56]:
```
xgb_model = XGBClassifier(objective="binary:logistic", random_state=123
, eval_metric="logloss")

random_search = RandomizedSearchCV(xgb_model, params, cv = 5)

random_search.fit(X_train_np, y_train)

print('Best hyperparameters:', random_search.best_params_)
```

```
Best hyperparameters: {'eta ': 0.08108581321624943, 'max_depth': 4}
```

In [57]:
```
best_params = {'eta ':0.08108581321624943, 'max_depth':4, 'eval_metric'
:'logloss', 'objective':'binary:logistic'}
```

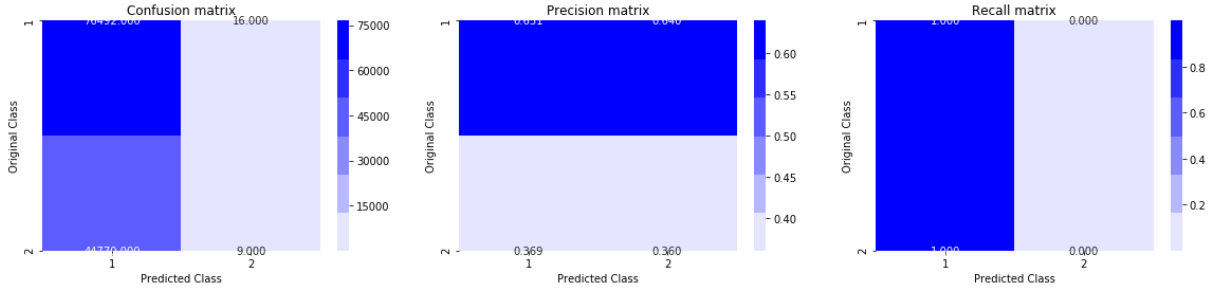
In [58]:
```
d_train = xgb.DMatrix(X_train_np, label=y_train)
d_test = xgb.DMatrix(X_test_np, label=y_test)

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

bst = xgb.train(best_params, d_train, 400, watchlist, early_stopping_ro
unds=20, verbose_eval=10)

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

```
[0]     train-logloss:0.677383  valid-logloss:0.675203
Multiple eval metrics have been passed: 'valid-logloss' will be used fo
r early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.659304  valid-logloss:0.658929
[20]    train-logloss:0.65763   valid-logloss:0.658774
Stopping. Best iteration:
[9]     train-logloss:0.658339  valid-logloss:0.658178
```

```
The test log loss is: 0.658707928161969
```

In [11]:
```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
Total number of data points : 121287
```



Summary

1. Handled the issue of data leakage by splitting training and testing data before performing vectorization.
2. Performed SVM and Logistic Regression on TF-IDF vectorization
3. After hypertuning xgboost by RandomizedSearchcv, the log-loss achieved is 0.658707928161969.