In [ ]:
```
![Quora.png](attachment:Quora.png)
```

Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

Problem Statement

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

1.2 Sources/Useful Links

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

Useful Links

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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.

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

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

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup

import warnings
warnings.filterwarnings("ignore")


import distance
from nltk.stem import PorterStemmer
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-pyt
hon3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image

import time
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
from tqdm import tqdm
# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from scipy.sparse import hstack
from collections import Counter
```

```
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import xgboost as xgb
import datetime as dt
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV,StratifiedKFold
```

3.1 Reading data and basic stats

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

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

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

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

Out[3]:

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

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

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

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

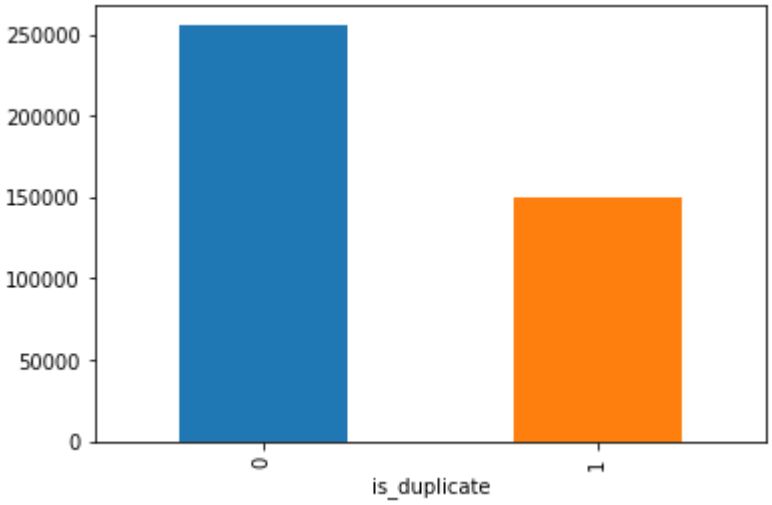
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

In [5]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x234739db198>



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

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

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

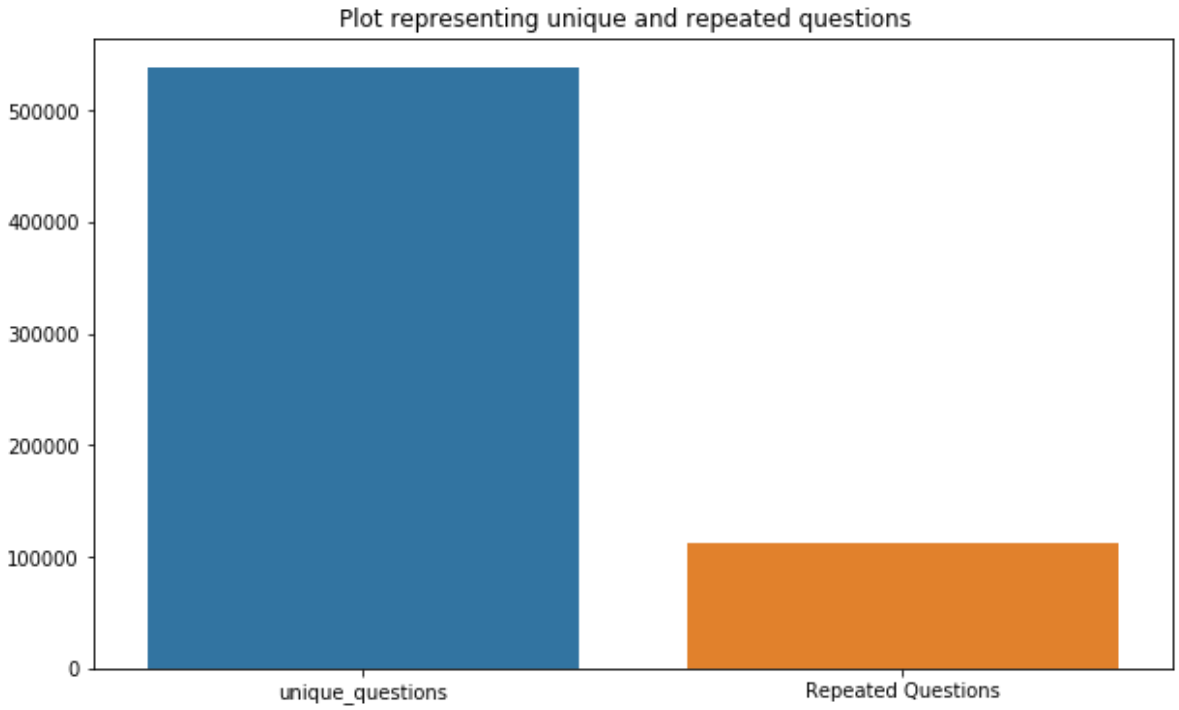
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.2 Number of unique questions

In [8]:
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

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

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.7795394
5937505%)

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

In [9]:
```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

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



3.2.3 Checking for Duplicates

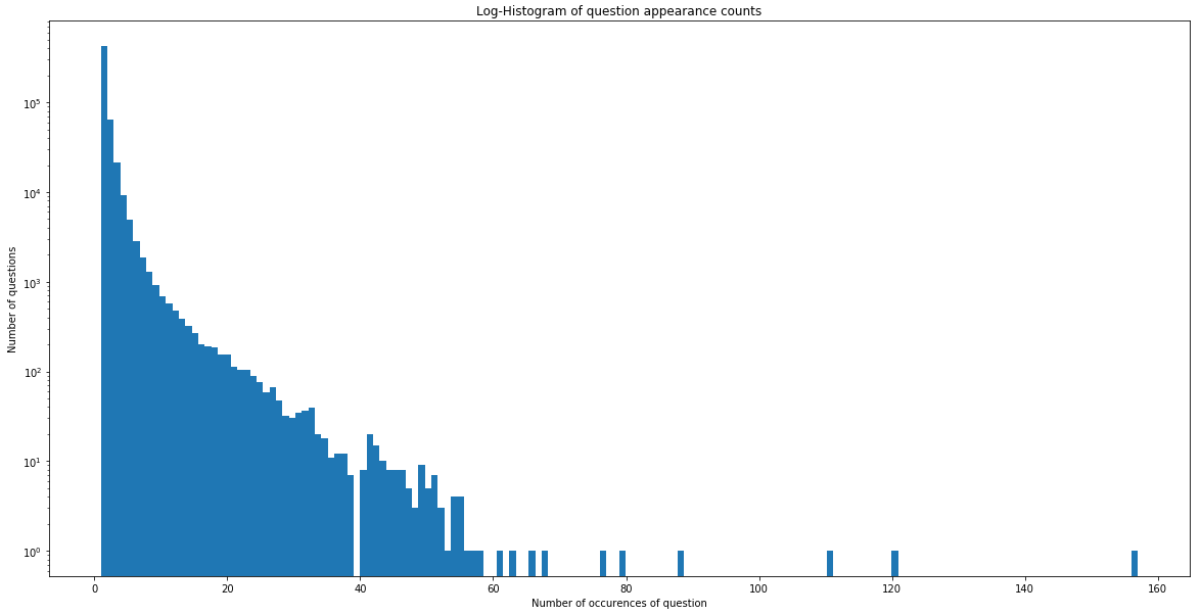
In [10]:
```
#checking whether there are any repeated pair of questions

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

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

```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [11]:
```
plt.figure(figsize=(20, 10))

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

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

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

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

plt.ylabel('Number of questions')

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

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



3.2.5 Checking for NULL values

In [12]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
df.info()
```

```
            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
<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
```

- There are two rows with null values in question2

In [13]:
```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
#df=df.sample(n=100000,random_state=1)
df.to_csv("train.csv")
df.shape
```

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

Out[13]: (404290, 6)

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

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

In [14]:
```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-
1')
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

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

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

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

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

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

df.head()
```

Out[14]:

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

3.3.1 Analysis of some of the extracted features

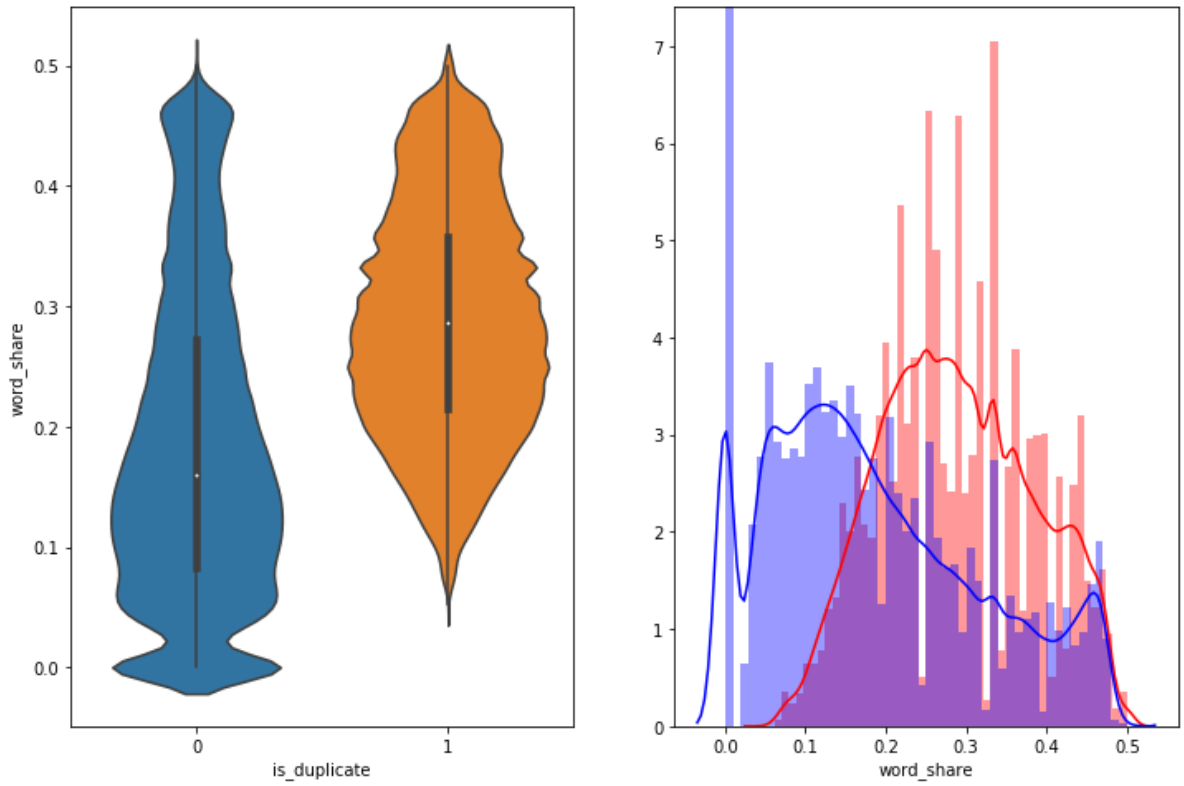
- Here are some questions have only one single words.

In [15]:
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words'
]))

print ("Minimum length of the questions in question2 : " , min(df['q2_n_words'
]))

print ("Number of Questions with minimum length [question1] :", df[df['q1_n_wo
rds']== 1].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_wo
rds']== 1].shape[0])
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

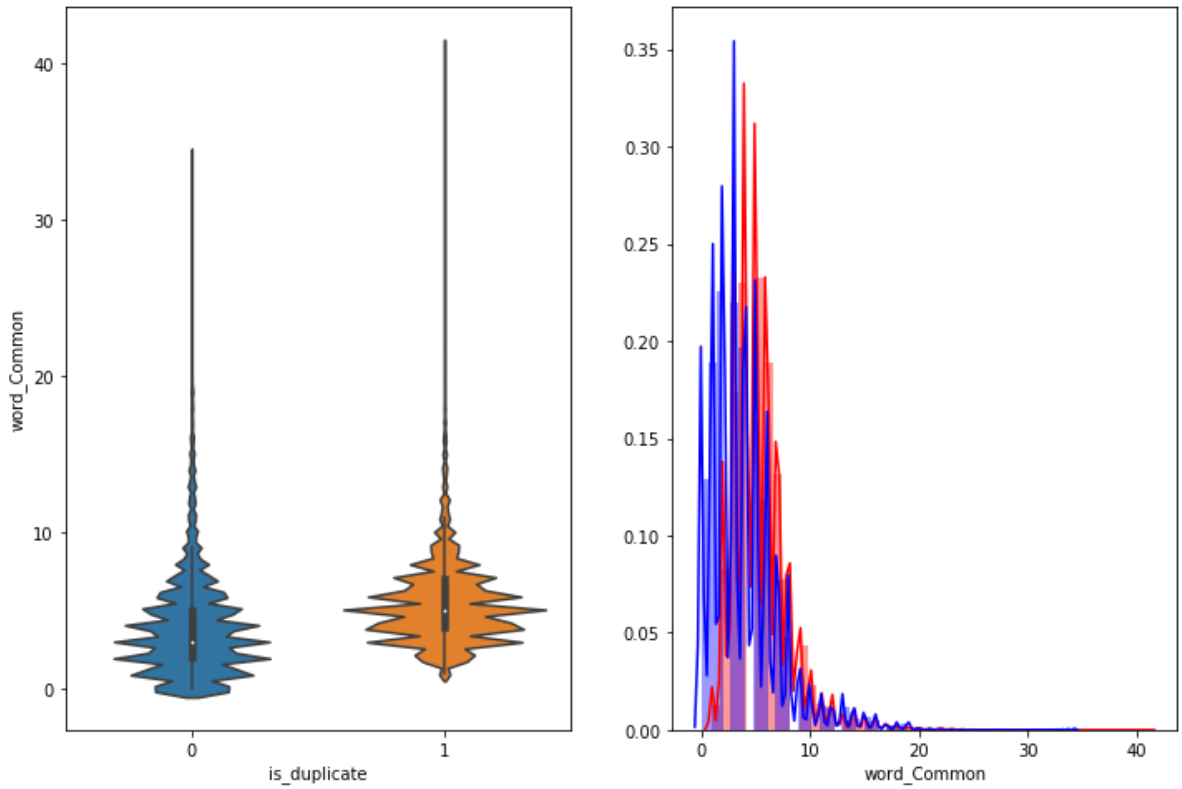
In [16]:
```
plt.figure(figsize=(12, 8))

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

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



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

3.3.1.2 Feature: word_Common

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

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

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



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

In [18]:
```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-ca
nt-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 pre
vious notebook")
```

In [19]:
```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
#df=df.sample(n=100000,random_state=1)
#df.to_csv("train.csv")
df.shape
```

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate, freq_qid1, freq
_qid2, q1len, q2len, q1_n_words, q2_n_words, word_Common, word_Total, word_sh
are, freq_q1+q2, freq_q1-q2]
Index: []
```

Out[19]: (404290, 17)

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

Out[20]:

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

3.4 Preprocessing of Text

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

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

STOP_WORDS = stopwords.words("english")


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


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

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


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


    return x
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [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_token
s)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_token
s)) + 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-func
tion-to-compare-2-strings
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["q
uestion1"], x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sort
ing the tokens alphabetically, and
    # then joining them back into a string We then compare the transformed str
ings 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["que
stion1"], 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_
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.99
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.59

2 rows × 21 columns

In [24]:
```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
#df=df.sample(n=100000,random_state=1)

df.shape
```

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate, cwc_min, cwc_ma
x, csc_min, csc_max, ctc_min, ctc_max, last_word_eq, first_word_eq, abs_len_d
iff, mean_len, token_set_ratio, token_sort_ratio, fuzz_ratio, fuzz_partial_ra
tio, longest_substr_ratio]
Index: []

[0 rows x 21 columns]
```

Out[24]: (404290, 21)

In [25]:
```
df[df.isnull().any(1)].shape
```

Out[25]: (0, 21)

In [26]:
```
df.fillna('', inplace=True)
```

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 [27]:
```
df_duplicate = df[df['is_duplicate'] == 1]
dfp_nonduplicate = df[df['is_duplicate'] == 0]

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

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

#Saving the np array into a text file
np.savetxt('train_p.txt', p, delimiter=' ', fmt='%s', 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
```

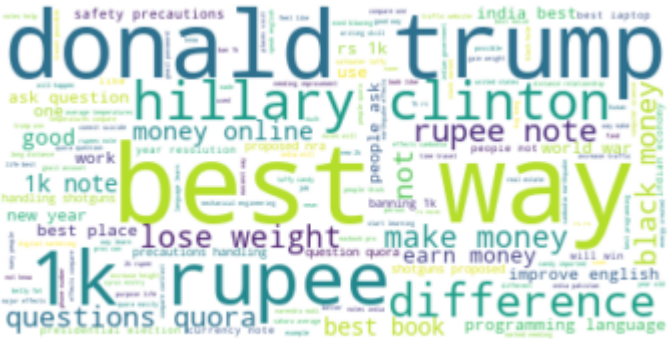
In [28]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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 : 33194832
```

Word Clouds generated from duplicate pair question's text

In [29]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=sto
pwords)
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

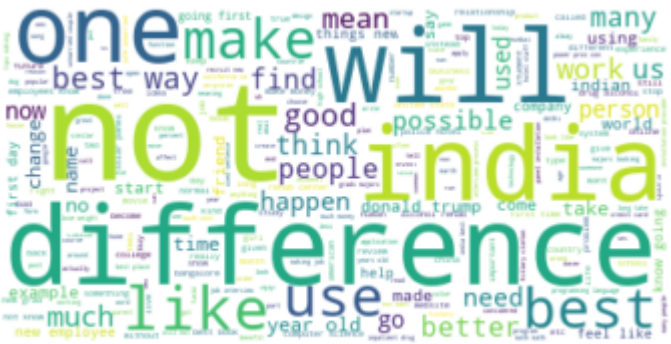
```
Word Cloud for Duplicate Question pairs
```



Word Clouds generated from non duplicate pair question's text

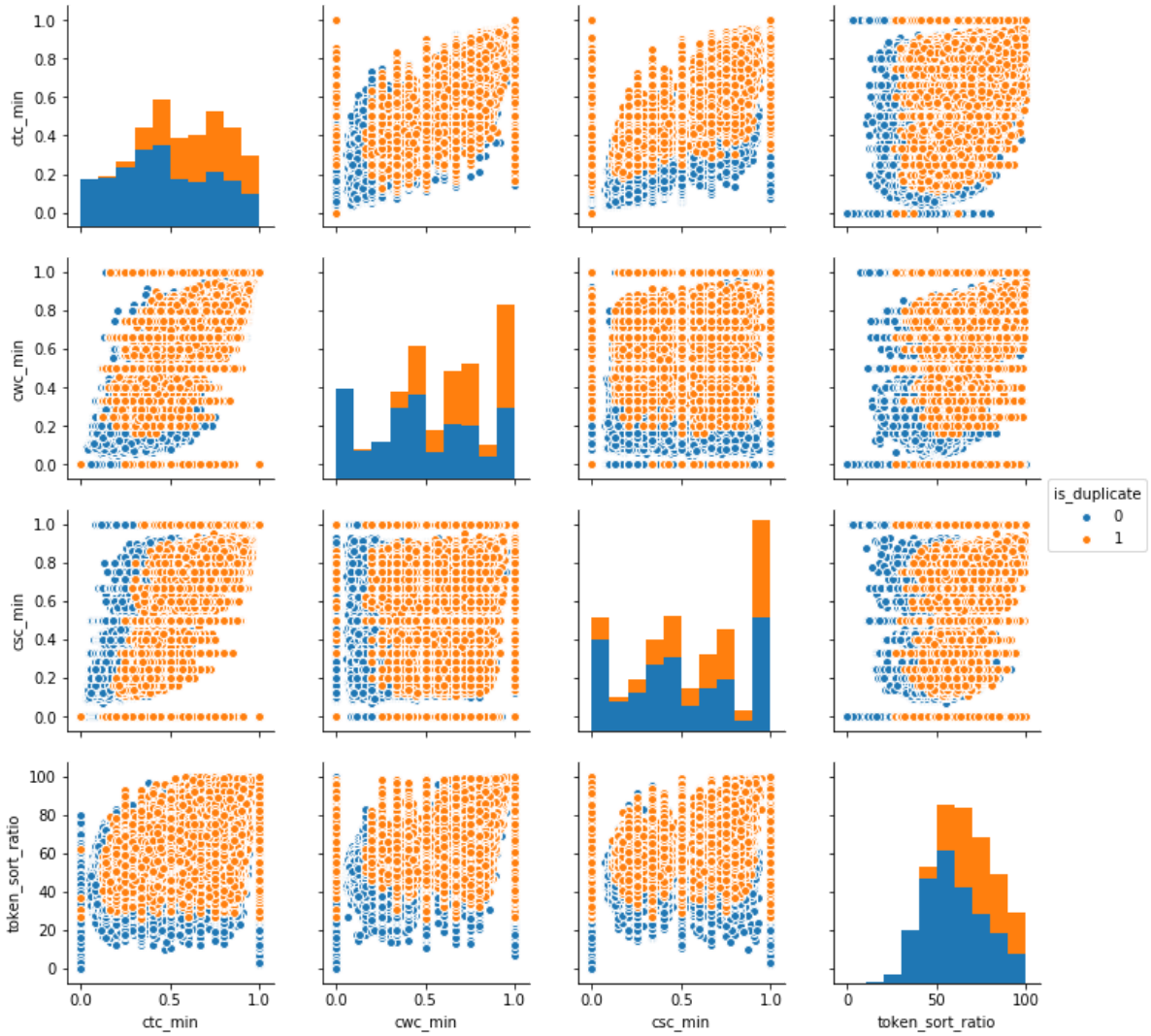
In [30]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stop
words)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

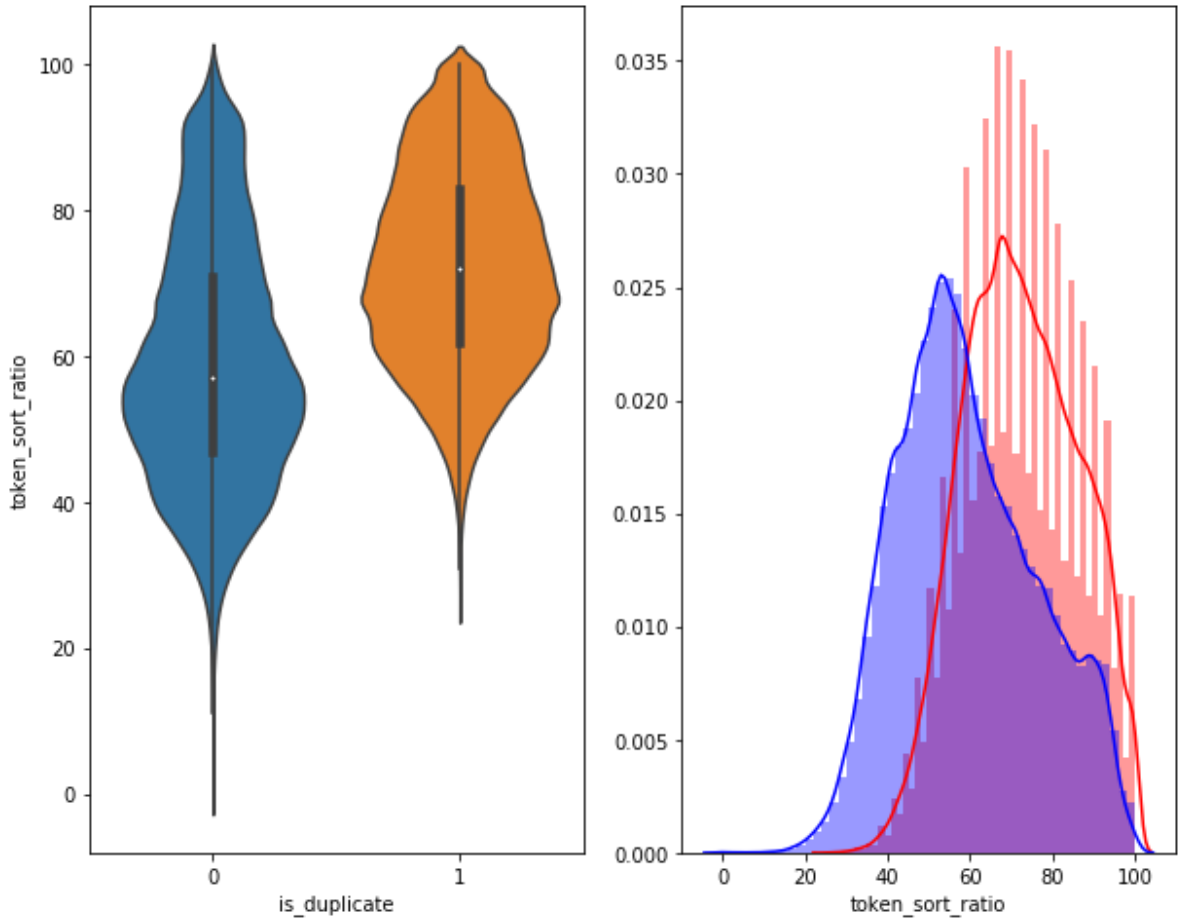


3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

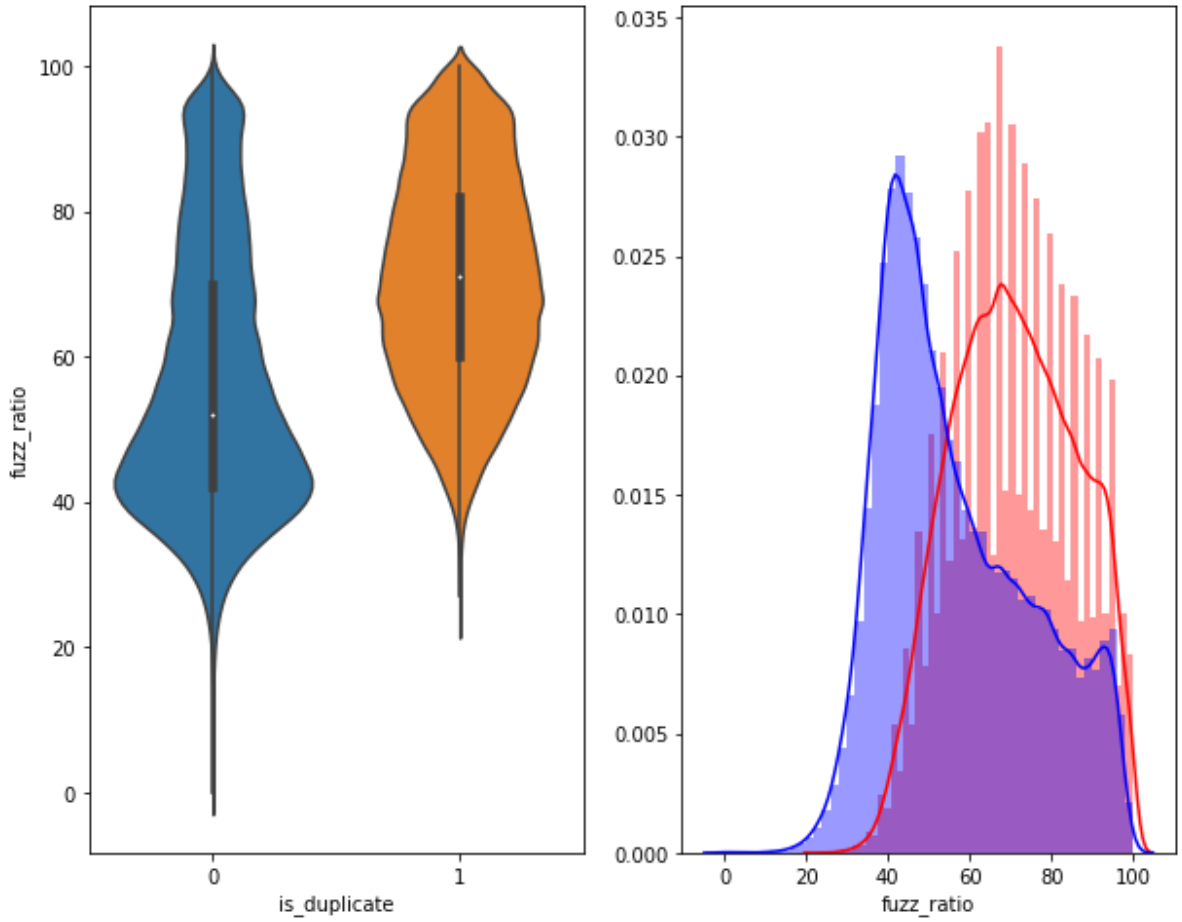
In [31]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_dupl
icate']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'tok
en_sort_ratio'])
plt.show()
```



In [32]:
```
# 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 [33]:
```
plt.figure(figsize=(10, 8))

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

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



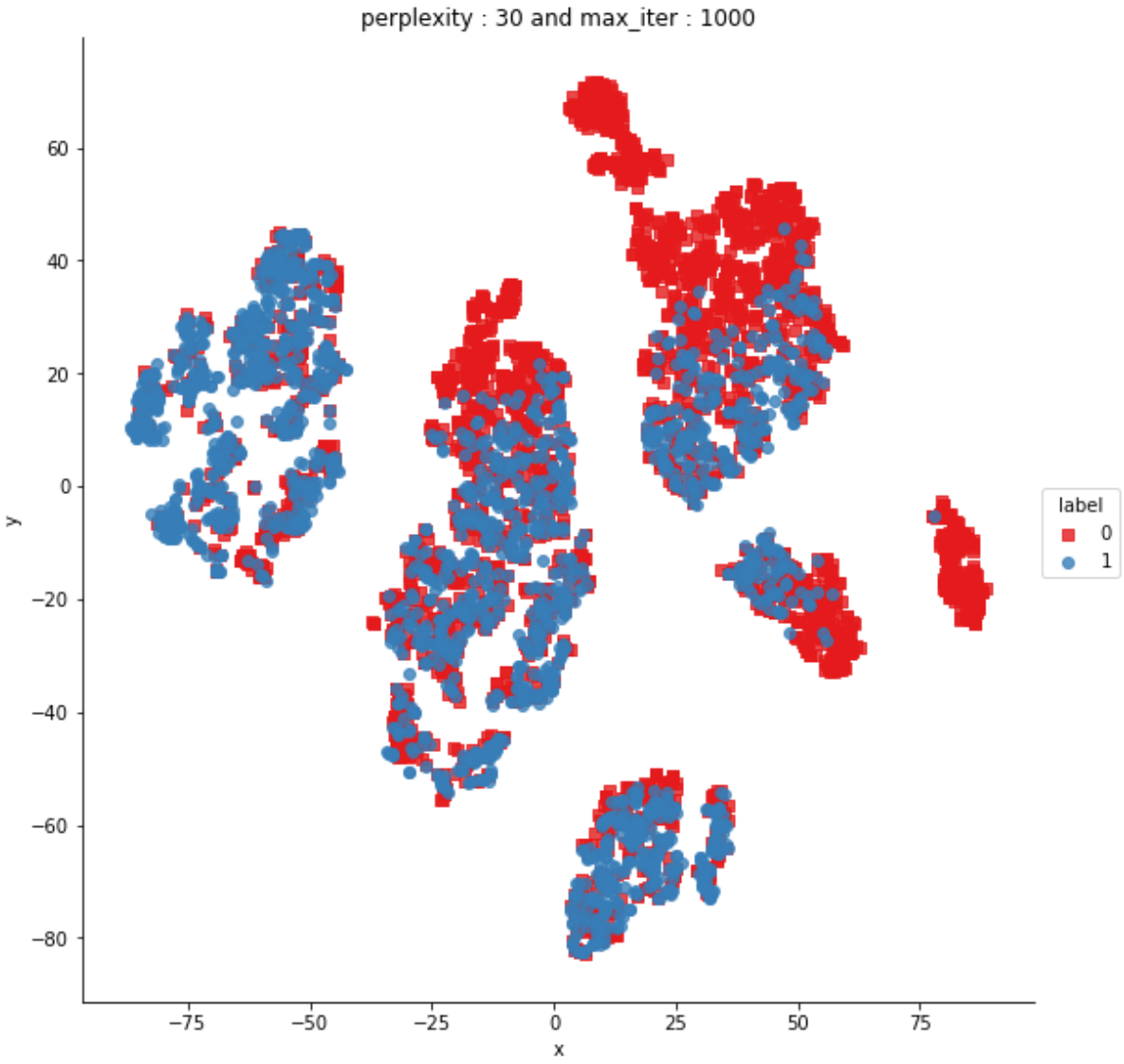
3.5.2 Visualization

In [34]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after clea
ning the data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

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

In [35]:
```
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 0.025s...
[t-SNE] Computed neighbors for 5000 samples in 0.788s...
[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 0.459s
[t-SNE] Iteration 50: error = 80.8968964, gradient norm = 0.0430571 (50 itera
tions in 13.308s)
[t-SNE] Iteration 100: error = 70.3833160, gradient norm = 0.0099593 (50 iter
ations in 10.015s)
[t-SNE] Iteration 150: error = 68.6159134, gradient norm = 0.0056708 (50 iter
ations in 10.144s)
[t-SNE] Iteration 200: error = 67.7694321, gradient norm = 0.0040581 (50 iter
ations in 10.886s)
[t-SNE] Iteration 250: error = 67.2746048, gradient norm = 0.0033067 (50 iter
ations in 10.944s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.274605
[t-SNE] Iteration 300: error = 1.7729300, gradient norm = 0.0011900 (50 itera
tions in 11.142s)
[t-SNE] Iteration 350: error = 1.3714967, gradient norm = 0.0004818 (50 itera
tions in 11.178s)
[t-SNE] Iteration 400: error = 1.2036748, gradient norm = 0.0002779 (50 itera
tions in 11.227s)
[t-SNE] Iteration 450: error = 1.1132656, gradient norm = 0.0001889 (50 itera
tions in 11.022s)
[t-SNE] Iteration 500: error = 1.0582460, gradient norm = 0.0001434 (50 itera
tions in 11.205s)
[t-SNE] Iteration 550: error = 1.0222589, gradient norm = 0.0001180 (50 itera
tions in 11.256s)
[t-SNE] Iteration 600: error = 0.9984865, gradient norm = 0.0001015 (50 itera
tions in 10.946s)
[t-SNE] Iteration 650: error = 0.9830498, gradient norm = 0.0000958 (50 itera
tions in 10.735s)
[t-SNE] Iteration 700: error = 0.9726909, gradient norm = 0.0000877 (50 itera
tions in 11.084s)
[t-SNE] Iteration 750: error = 0.9647216, gradient norm = 0.0000823 (50 itera
tions in 11.431s)
[t-SNE] Iteration 800: error = 0.9582971, gradient norm = 0.0000755 (50 itera
tions in 11.119s)
[t-SNE] Iteration 850: error = 0.9531373, gradient norm = 0.0000697 (50 itera
tions in 10.910s)
[t-SNE] Iteration 900: error = 0.9484153, gradient norm = 0.0000696 (50 itera
tions in 10.727s)
[t-SNE] Iteration 950: error = 0.9445393, gradient norm = 0.0000659 (50 itera
tions in 10.507s)
[t-SNE] Iteration 1000: error = 0.9412127, gradient norm = 0.0000674 (50 iter
ations in 10.563s)
[t-SNE] Error after 1000 iterations: 0.941213
```

In [36]:
```
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 [37]:
```
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 0.020s...
[t-SNE] Computed neighbors for 5000 samples in 0.768s...
[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 0.512s
[t-SNE] Iteration 50: error = 80.3592682, gradient norm = 0.0335202 (50 itera
tions in 25.281s)
[t-SNE] Iteration 100: error = 69.1112671, gradient norm = 0.0036575 (50 iter
ations in 13.155s)
[t-SNE] Iteration 150: error = 67.6171112, gradient norm = 0.0017708 (50 iter
ations in 12.193s)
[t-SNE] Iteration 200: error = 67.0565109, gradient norm = 0.0011567 (50 iter
ations in 11.459s)
[t-SNE] Iteration 250: error = 66.7296524, gradient norm = 0.0009161 (50 iter
ations in 12.053s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729652
[t-SNE] Iteration 300: error = 1.4983541, gradient norm = 0.0006807 (50 itera
tions in 15.329s)
[t-SNE] Iteration 350: error = 1.1549147, gradient norm = 0.0001922 (50 itera
tions in 17.709s)
[t-SNE] Iteration 400: error = 1.0101781, gradient norm = 0.0000912 (50 itera
tions in 17.479s)
[t-SNE] Iteration 450: error = 0.9388669, gradient norm = 0.0000628 (50 itera
tions in 18.776s)
[t-SNE] Iteration 500: error = 0.9029322, gradient norm = 0.0000524 (50 itera
tions in 17.634s)
[t-SNE] Iteration 550: error = 0.8841860, gradient norm = 0.0000482 (50 itera
tions in 18.104s)
[t-SNE] Iteration 600: error = 0.8722453, gradient norm = 0.0000365 (50 itera
tions in 17.206s)
[t-SNE] Iteration 650: error = 0.8627461, gradient norm = 0.0000347 (50 itera
tions in 17.161s)
[t-SNE] Iteration 700: error = 0.8549610, gradient norm = 0.0000312 (50 itera
tions in 16.632s)
[t-SNE] Iteration 750: error = 0.8487639, gradient norm = 0.0000311 (50 itera
tions in 17.424s)
[t-SNE] Iteration 800: error = 0.8440317, gradient norm = 0.0000281 (50 itera
tions in 17.571s)
[t-SNE] Iteration 850: error = 0.8396705, gradient norm = 0.0000250 (50 itera
tions in 19.219s)
[t-SNE] Iteration 900: error = 0.8354425, gradient norm = 0.0000242 (50 itera
tions in 18.440s)
[t-SNE] Iteration 950: error = 0.8317489, gradient norm = 0.0000233 (50 itera
tions in 17.074s)
[t-SNE] Iteration 1000: error = 0.8288577, gradient norm = 0.0000257 (50 iter
ations in 17.284s)
[t-SNE] Error after 1000 iterations: 0.828858
```

In [38]:
```
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 feature
s')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

In [39]:
```
# 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 [40]:
```
df.head(2)
```

Out[40]:

	Unnamed: 0	id	qid1	qid2	question1	question2	is_duplicate
0	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	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0

In [41]:
```
#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 noteboo
k")

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='lat
in-1', nrows=100000)
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run pr
evious notebook")
```

In [42]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1
)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=
1)
```

In [43]:
```
df3 = dfnlp[['id','question1','question2']]
duplicate = dfnlp.is_duplicate
```

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

Out[44]: pandas.core.series.Series

In [45]:
```
#df1=df1.drop('Unnamed: 0',axis=1)
df3 = df3.fillna(' ')
#assigning new dataframe with columns question(q1+q2) and id same as df3
new_df = pd.DataFrame()
new_df['questions'] = df3.question1 + ' ' + df3.question2
new_df['id'] = df3.id
df2['id']=df1['id']
new_df['id']=df1['id']
final_df = df1.merge(df2, on='id',how='left') #merging df1 and df2


#final_df=final_df.sample(n=100000,random_state=1)
final_df = final_df.fillna('')
nan_rows = final_df[final_df.isnull().any(1)]
print (nan_rows)
#df=df.sample(n=100000,random_state=1)

final_df.shape

X  = final_df.merge(new_df, on='id',how='left')#merging final_df and new_df
```

```
Empty DataFrame
Columns: [id, cwc_min, cwc_max, csc_min, csc_max, ctc_min, ctc_max, last_word
_eq, first_word_eq, abs_len_diff, mean_len, token_set_ratio, token_sort_rati
o, fuzz_ratio, fuzz_partial_ratio, longest_substr_ratio, freq_qid1, freq_qid
2, q1len, q2len, q1_n_words, q2_n_words, word_Common, word_Total, word_share,
freq_q1+q2, freq_q1-q2]
Index: []

[0 rows x 27 columns]
```

In [46]:
```
#removing id from X
X=X.drop('id',axis=1)
X.columns
```

Out[46]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [47]:
```
type(X)
```

Out[47]:
```
pandas.core.frame.DataFrame
```

In [48]:
```
y=np.array(duplicate)
```

In [49]:
```
#y= y[0:100000]
```

In [50]:
```
X.shape
```

Out[50]: (404290, 27)

In [51]:
```
y.shape
```

Out[51]: (404290,)

Random train test split(70:30)

In [52]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=3,test_size=0.3)
```

In [53]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

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

In [54]:
```
#seperating questions for tfidf vectorizer
X_train_ques=X_train['questions']
X_test_ques=X_test['questions']

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

In [55]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts

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

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

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

In [56]:
```
# 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(X_train_ques)):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
#df['q1_feats_m'] = list(vecs1)
```

```
100%|████████████████████████████████████████████████████████████████████████
| 283003/283003 [1:20:59<00:00, 58.24it/s]
```

In [57]:
```
vecs2 = []
for qu2 in tqdm(list(X_test_ques)):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 96])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
#df['q2_feats_m'] = list(vecs2)
```

```
100%|████████████████████████████████████████████████████████████████████████
██| 121287/121287 [35:13<00:00, 57.39it/s]
```

In [58]:
```
first_df=pd.DataFrame(vecs1)
sec_df=pd.DataFrame(vecs2)
```

In [59]:
```
from scipy.sparse import hstack
X_train = hstack((X_train.values,first_df), format='csr', dtype='float64')
X_test= hstack((X_test.values,sec_df), format='csr', dtype='float64')
print(X_train.shape)
print(X_test.shape)
```

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

In [60]:
```
type(X_train)
```

Out[60]:
```
scipy.sparse.csr.csr_matrix
```

4. Machine Learning Models

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

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

In [62]:
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1
])/train_len)
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/
test_len)
```

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

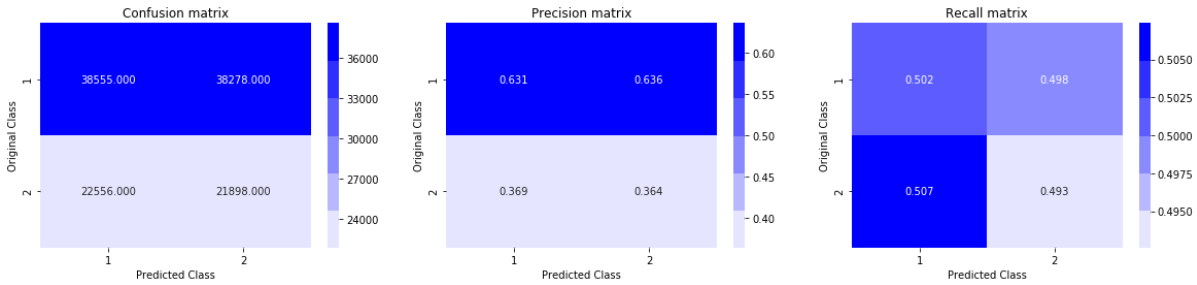
In [63]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at 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, ytick
labels=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, ytick
labels=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, ytick
```

```
labels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

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

In [64]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
r 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 0.8933834447842192
```



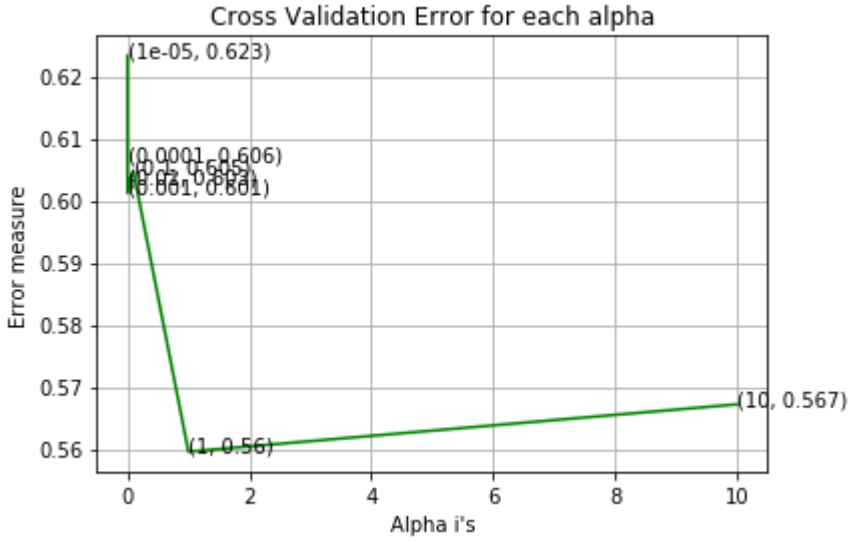
In [65]:
```
type(y_train)
```

Out[65]:
```
numpy.ndarray
```

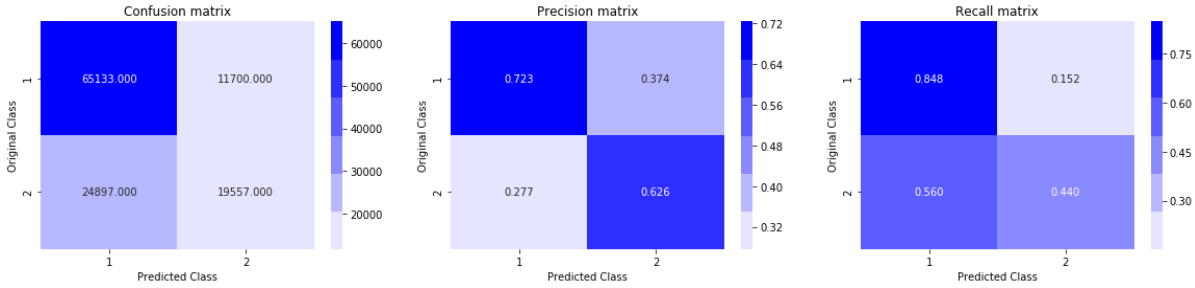
4.4 Logistic Regression with hyperparameter tuning

In [66]:
```
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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))

#------Cross Validation Error for each alpha------------------
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()

#-------------------Fitting the model to the best alpha value that is obtained
--------------------
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 i
s:",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 i
s:",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: 0.6233485931159208
For values of alpha =  0.0001 The log loss is: 0.6064437319566944
For values of alpha =  0.001 The log loss is: 0.6012018853511233
For values of alpha =  0.01 The log loss is: 0.6027785847299325
For values of alpha =  0.1 The log loss is: 0.6045981300630434
For values of alpha =  1 The log loss is: 0.5595746326358305
For values of alpha =  10 The log loss is: 0.5671879617922797
```



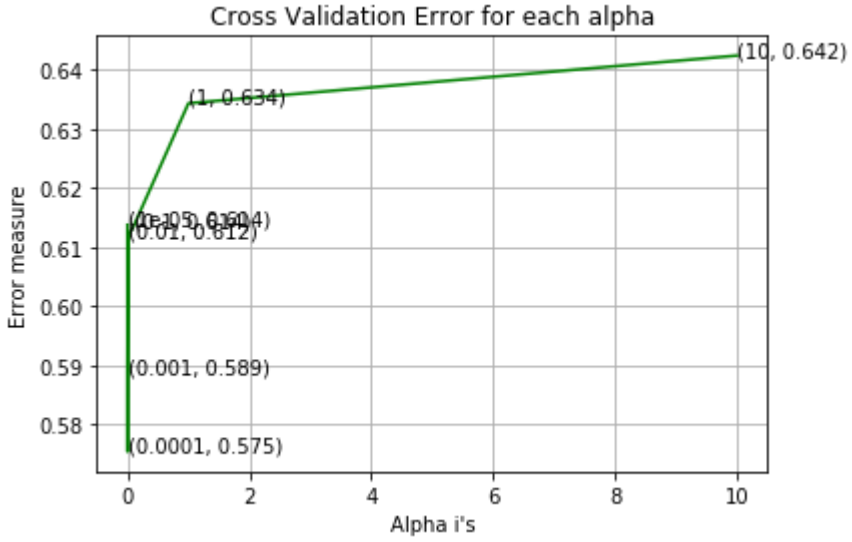

```
For values of best alpha =  1 The train log loss is: 0.5578025535690638
For values of best alpha =  1 The test log loss is: 0.5595746326358305
Total number of data points : 121287
```



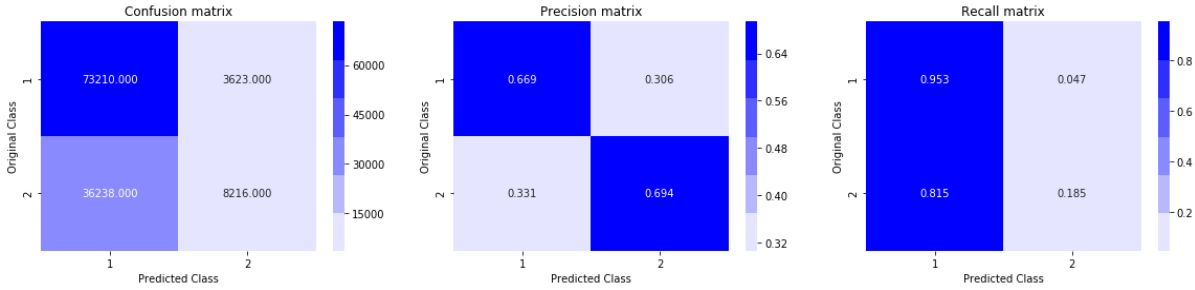
4.5 Linear SVM with hyperparameter tuning

In [67]:
```
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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))

#------Cross Validation Error for each alpha-------------------
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()

#-------------------Fitting the model to the best alpha value that is obtained
--------------------
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', rando
m_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 i
s:",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 i
s:",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: 0.6137602935788894
For values of alpha =  0.0001 The log loss is: 0.575388881152981
For values of alpha =  0.001 The log loss is: 0.5887301721242333
For values of alpha =  0.01 The log loss is: 0.6118495961745873
For values of alpha =  0.1 The log loss is: 0.6135207158732119
For values of alpha =  1 The log loss is: 0.6343249151865272
For values of alpha =  10 The log loss is: 0.6423945800604853
```




```
For values of best alpha =  0.0001 The train log loss is: 0.575636981605262
For values of best alpha =  0.0001 The test log loss is: 0.575388881152981
Total number of data points : 121287
```



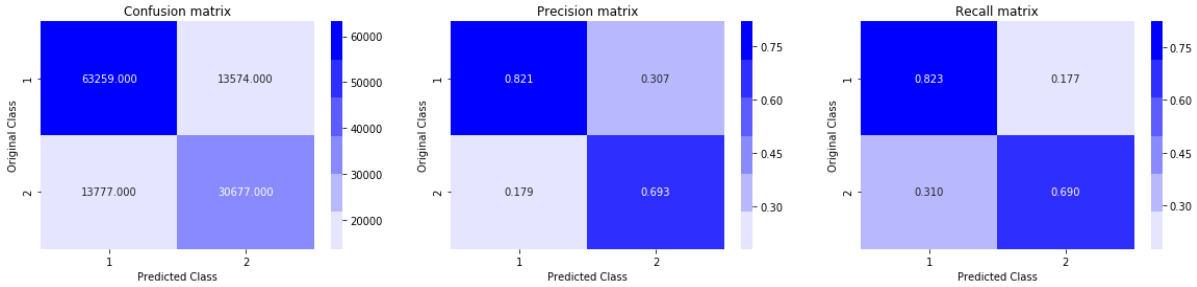
4.6 XGBoost

In [68]:
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

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

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

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

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

```
[0]     train-logloss:0.686855  valid-logloss:0.686897
Multiple eval metrics have been passed: 'valid-logloss' will be used for earl
y stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.635175  valid-logloss:0.635201
[20]    train-logloss:0.598585  valid-logloss:0.598467
[30]    train-logloss:0.570867  valid-logloss:0.570639
[40]    train-logloss:0.549784  valid-logloss:0.549599
[50]    train-logloss:0.532992  valid-logloss:0.533008
[60]    train-logloss:0.519763  valid-logloss:0.519805
[70]    train-logloss:0.509172  valid-logloss:0.509103
[80]    train-logloss:0.500515  valid-logloss:0.500578
[90]    train-logloss:0.49332   valid-logloss:0.493421
[100]   train-logloss:0.487305  valid-logloss:0.487403
[110]   train-logloss:0.482302  valid-logloss:0.482498
[120]   train-logloss:0.478175  valid-logloss:0.478449
[130]   train-logloss:0.474337  valid-logloss:0.474681
[140]   train-logloss:0.470921  valid-logloss:0.471442
[150]   train-logloss:0.467981  valid-logloss:0.468608
[160]   train-logloss:0.46527   valid-logloss:0.466009
[170]   train-logloss:0.462744  valid-logloss:0.463588
[180]   train-logloss:0.46047   valid-logloss:0.461443
[190]   train-logloss:0.458351  valid-logloss:0.45941
[200]   train-logloss:0.45658   valid-logloss:0.457717
[210]   train-logloss:0.454804  valid-logloss:0.45605
[220]   train-logloss:0.45322   valid-logloss:0.454556
[230]   train-logloss:0.45153   valid-logloss:0.452928
[240]   train-logloss:0.449865  valid-logloss:0.451382
[250]   train-logloss:0.448233  valid-logloss:0.449862
[260]   train-logloss:0.446724  valid-logloss:0.448448
[270]   train-logloss:0.445484  valid-logloss:0.447317
[280]   train-logloss:0.44401   valid-logloss:0.445942
[290]   train-logloss:0.442866  valid-logloss:0.444894
[300]   train-logloss:0.441668  valid-logloss:0.443807
[310]   train-logloss:0.440532  valid-logloss:0.442774
[320]   train-logloss:0.439443  valid-logloss:0.441769
[330]   train-logloss:0.438515  valid-logloss:0.440926
[340]   train-logloss:0.437537  valid-logloss:0.440033
[350]   train-logloss:0.436608  valid-logloss:0.439176
[360]   train-logloss:0.435651  valid-logloss:0.438308
[370]   train-logloss:0.434678  valid-logloss:0.437427
[380]   train-logloss:0.433823  valid-logloss:0.436669
[390]   train-logloss:0.432868  valid-logloss:0.435825
[399]   train-logloss:0.43206   valid-logloss:0.435093
The test log loss is: 0.4350925441502612
```

In [69]:
```
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
```



XgBoost models using RandomsearchCV with vectorizer as TF-IDF W2V

In [70]:
```
def hyperparameter_tunning(X,Y):
    params = {'n_estimators' : [10, 20, 40, 60, 80, 100, 120, 150],'learning_r
ate' :[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]}
    param_grid = params

    model = XGBClassifier(nthread=-1)
    kfold = StratifiedKFold(n_splits=5, shuffle=True)
    random_search = RandomizedSearchCV(model, param_grid, scoring="neg_log_los
s", 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 [71]:
```
start = dt.datetime.now()

# Tune hyperparameter values
random_result = hyperparameter_tunning(X_train, y_train)

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

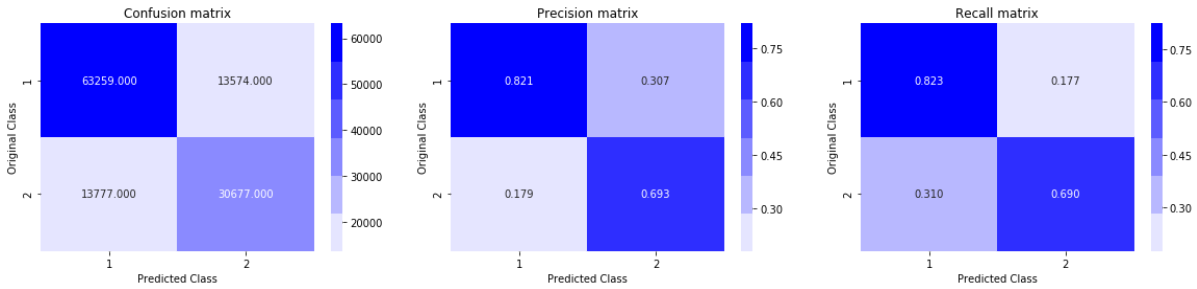
```
Best: -0.415893 using {'n_estimators': 120, 'learning_rate': 0.3}

-0.690259 (0.000019) with: {'n_estimators': 100, 'learning_rate': 0.0001}
-0.456645 (0.001476) with: {'n_estimators': 60, 'learning_rate': 0.1}
-0.609476 (0.000555) with: {'n_estimators': 40, 'learning_rate': 0.01}
-0.498611 (0.000917) with: {'n_estimators': 10, 'learning_rate': 0.2}
-0.415893 (0.001633) with: {'n_estimators': 120, 'learning_rate': 0.3}
-0.692565 (0.000004) with: {'n_estimators': 20, 'learning_rate': 0.0001}
-0.689689 (0.000023) with: {'n_estimators': 120, 'learning_rate': 0.0001}
-0.425572 (0.001961) with: {'n_estimators': 100, 'learning_rate': 0.2}
-0.563246 (0.000880) with: {'n_estimators': 80, 'learning_rate': 0.01}
-0.671542 (0.000147) with: {'n_estimators': 80, 'learning_rate': 0.001}

TimeTaken:  5:44:33.805972
```

In [72]:
```
start = dt.datetime.now()
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

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

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

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

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

```
The test log loss is: 0.4350925441502612
```

In [73]:
```
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
```



Perform For TFIDF features

In [74]:
```
dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = dfnlp[['id','question1','question2']]
duplicate = dfnlp.is_duplicate
```

In [75]:
```
#df1=df1.drop('Unnamed: 0',axis=1)
df3 = df3.fillna(' ')
#assigning new dataframe with columns question(q1+q2) and id same as df3
new_df = pd.DataFrame()
new_df['questions'] = df3.question1 + ' ' + df3.question2
new_df['id'] = df3.id
df2['id']=df1['id']
new_df['id']=df1['id']
final_df = df1.merge(df2, on='id',how='left') #merging df1 and df2
X  = final_df.merge(new_df, on='id',how='left')#merging final_df and new_df
```

In [76]:
```
#removing id from X
X=X.drop('id',axis=1)
X.columns
```

Out[76]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [77]:
```
y=np.array(duplicate)
```

Splitted the dataset into train and test (70:30)

In [78]:
```
#splitting data into train and test
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=3,test_size=0.
3)
```

In [79]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

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

In [80]:
```
#seperating questions for tfidf vectorizer
X_train_ques=X_train['questions']
X_test_ques=X_test['questions']

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

In [81]:
```
#tfidf vectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,3),min_df=10)
X_train_tfidf=tf_idf_vect.fit_transform(X_train_ques)
X_test_tfidf=tf_idf_vect.transform(X_test_ques)
```

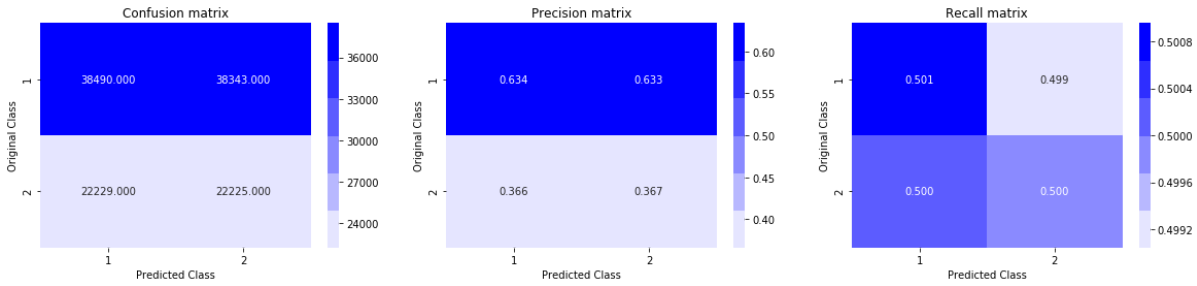
In [82]:
```
#adding tfidf features to our train and test data using hstack
X_train = hstack((X_train.values,X_train_tfidf))
X_test= hstack((X_test.values,X_test_tfidf))
print(X_train.shape)
print(X_test.shape)
```

```
(283003, 122688)
(121287, 122688)
```

Random Model

In [87]:
```
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)
```

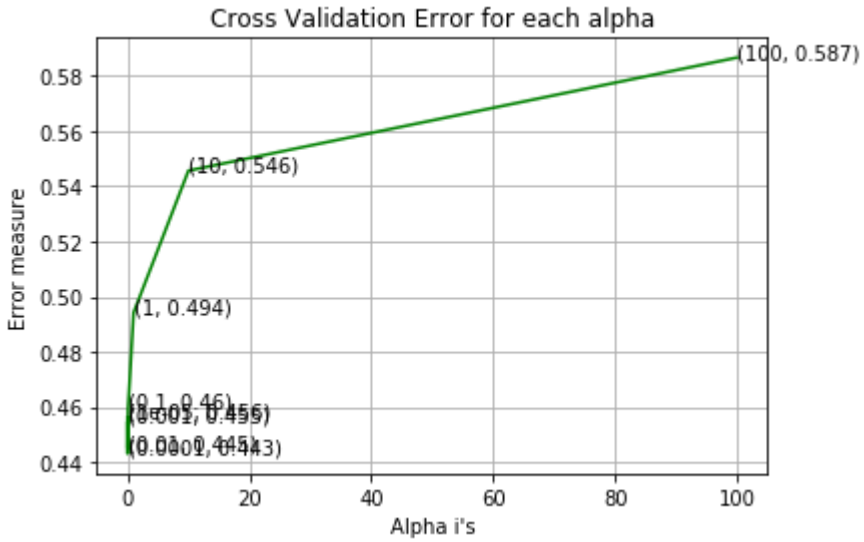
```
Log loss on Test Data using Random Model 0.8839553654779079
```



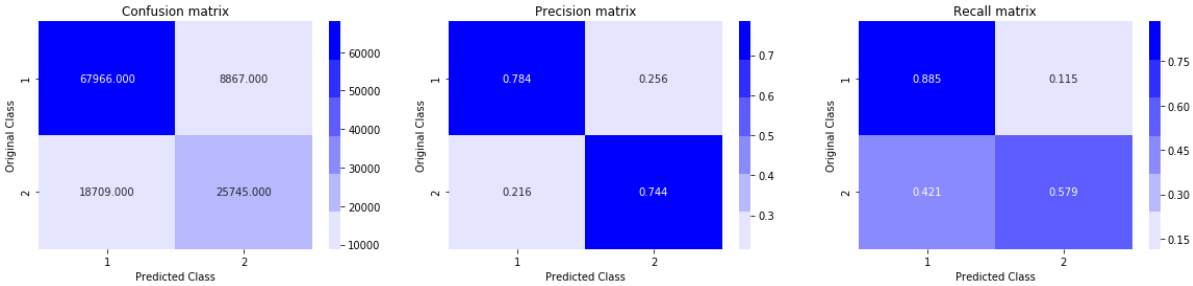
Applying Logistic Regression

In [83]:
```
alpha = [10 ** x for x in range(-5, 3)] # 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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))

#------Cross Validation Error for each alpha------------------
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()

#-------------------Fitting the model to the best alpha value that is obtained
--------------------
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 i
s:",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 i
s:",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: 0.4562485965456867
For values of alpha =  0.0001 The log loss is: 0.44306955033138756
For values of alpha =  0.001 The log loss is: 0.454674492788191
For values of alpha =  0.01 The log loss is: 0.4450557517762759
For values of alpha =  0.1 The log loss is: 0.45965583531614657
For values of alpha =  1 The log loss is: 0.49418254376282117
For values of alpha =  10 The log loss is: 0.5456857233802797
For values of alpha =  100 The log loss is: 0.5866835739439517
```



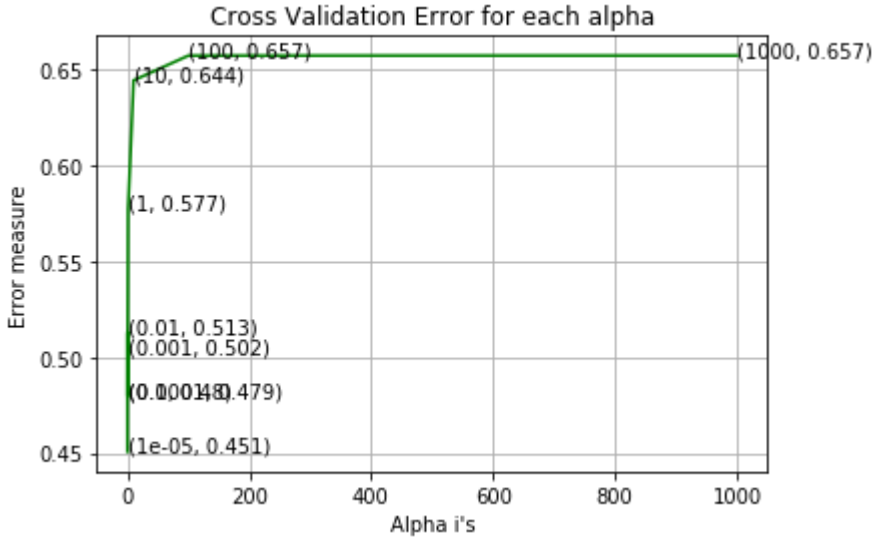

```
For values of best alpha =  0.0001 The train log loss is: 0.44473131588898546
For values of best alpha =  0.0001 The test log loss is: 0.44306955033138756
Total number of data points : 121287
```



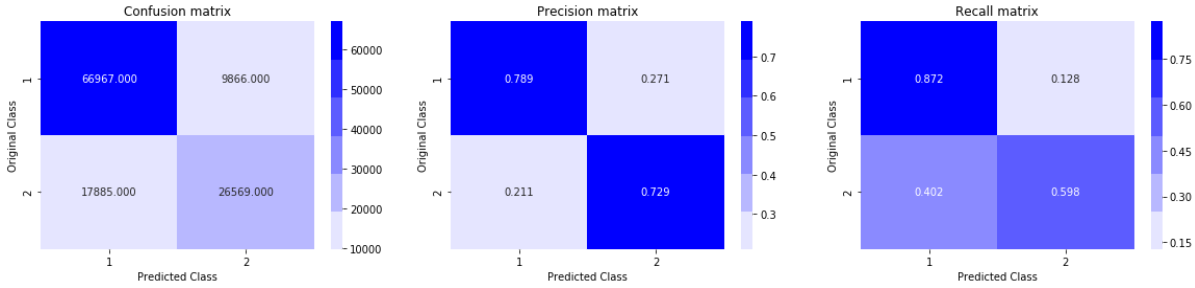
Applying Linear SVM

In [84]:
```
alpha = [10 ** x for x in range(-5, 4)] # 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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))

#------Cross Validation Error for each alpha------------------
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()

#-------------------Fitting the model to the best alpha value that is obtained
--------------------
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', rando
m_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 i
s:",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 i
s:",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: 0.4507465909306012
For values of alpha =  0.0001 The log loss is: 0.47943501857268417
For values of alpha =  0.001 The log loss is: 0.502288257227882
For values of alpha =  0.01 The log loss is: 0.5128240782666506
For values of alpha =  0.1 The log loss is: 0.4796289872710096
For values of alpha =  1 The log loss is: 0.576910408176294
For values of alpha =  10 The log loss is: 0.6441932149562888
For values of alpha =  100 The log loss is: 0.6571084989895937
For values of alpha =  1000 The log loss is: 0.6571084989603588
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4514720455851589
For values of best alpha =  1e-05 The test log loss is: 0.4507465909306012
Total number of data points : 121287
```



Conclusion

In [88]:
```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "

ptable.add_row(["******","Tokenizer - TFIDF W2V","******"])
ptable.add_row([">>>>>>","Dataset Size - 400K Points",">>>>>>"])

ptable.field_names = ['Model Name', 'Hyperparameter Tunning', 'Test Log Loss']
ptable.add_row(["Random","NA","0.893"])
ptable.add_row(["Logistic Regression","Done","0.559"])
ptable.add_row(["Linear SVM","Done","0.575"])
ptable.add_row(["XGBoost","NA","0.435"])
ptable.add_row(["XGBoost","Done","0.435"])

ptable.add_row(["\n","\n","\n"])
ptable.add_row(["******","Tokenizer - TFIDF","******"])

ptable.add_row(["Random","NA","0.883"])
ptable.add_row(["Logistic Regression","Done","0.443"])
ptable.add_row(["Linear SVM","Done","0.450"])

print(ptable)
```

```
+---------------------+----------------------------+---------------+
|      Model Name     |   Hyperparameter Tunning   | Test Log Loss |
+---------------------+----------------------------+---------------+
|        ******       |   Tokenizer - TFIDF W2V    |     ******    |
|        >>>>>>       | Dataset Size - 400K Points |     >>>>>>    |
|        Random       |             NA             |     0.893     |
| Logistic Regression |            Done            |     0.559     |
|      Linear SVM     |            Done            |     0.575     |
|       XGBoost       |             NA             |     0.435     |
|       XGBoost       |            Done            |     0.435     |
|                     |                            |               |
|                     |                            |               |
|        ******       |     Tokenizer - TFIDF      |     ******    |
|        Random       |             NA             |     0.883     |
| Logistic Regression |            Done            |     0.443     |
|      Linear SVM     |            Done            |     0.450     |
+---------------------+----------------------------+---------------+
```