Quora-1.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

2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share
market in india?","What is the step by step guide to invest in
share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamon
d?","What would happen if the Indian government stole the Kohi
noor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geologist?","What should I
do to be a great geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","
How can I see all my Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

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 [7]:
```
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
```

3.1 Reading data and basic stats

In [8]:
```
df = pd.read_csv("train.csv")

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

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

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

Out[9]:

	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 [10]:
```
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:

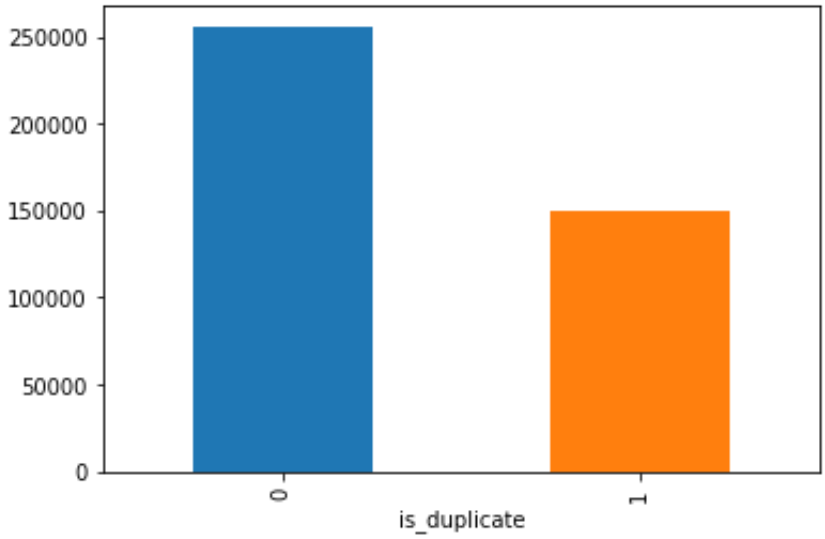
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

In [7]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1fd52fd0>
```



In [8]:
```
print('~> Total number of question pairs for training:\n   {}'.format(
```

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

In [9]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.fo
```

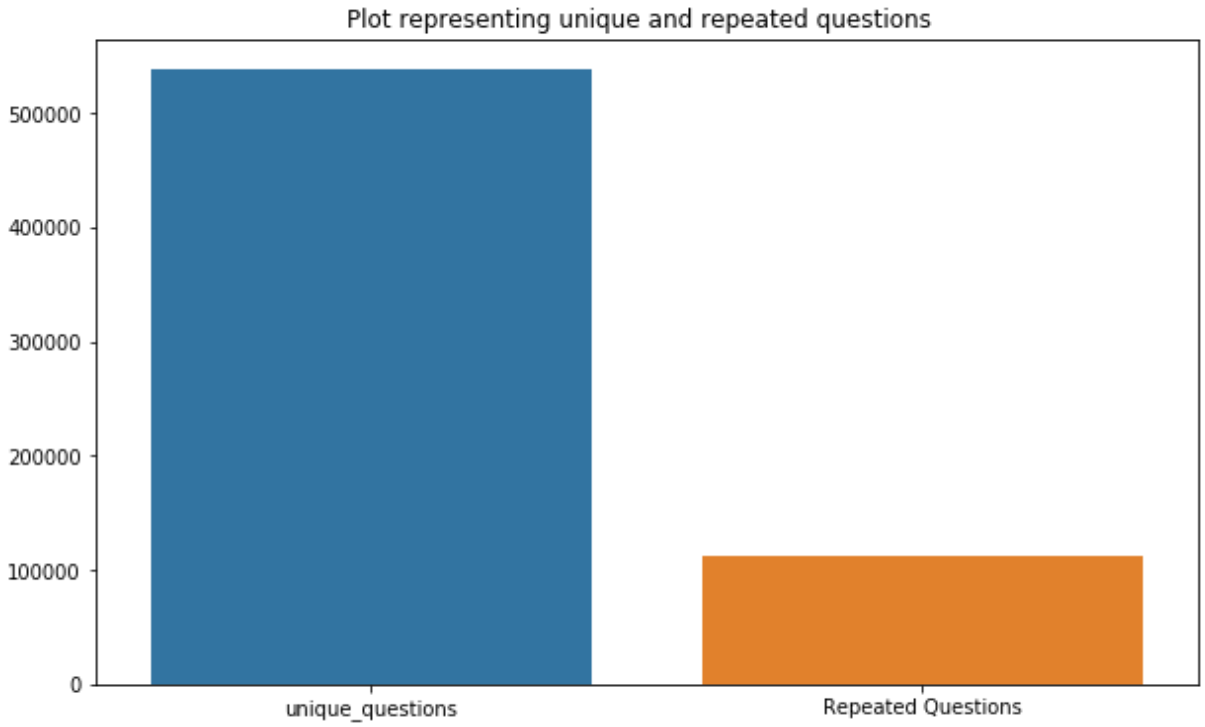
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.2 Number of unique questions

In [10]:
```
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: {}

print ('Max number of times a single question is repeated: {}\n'.forma

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 (2
0.77953945937505%)

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

In [11]:
```
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

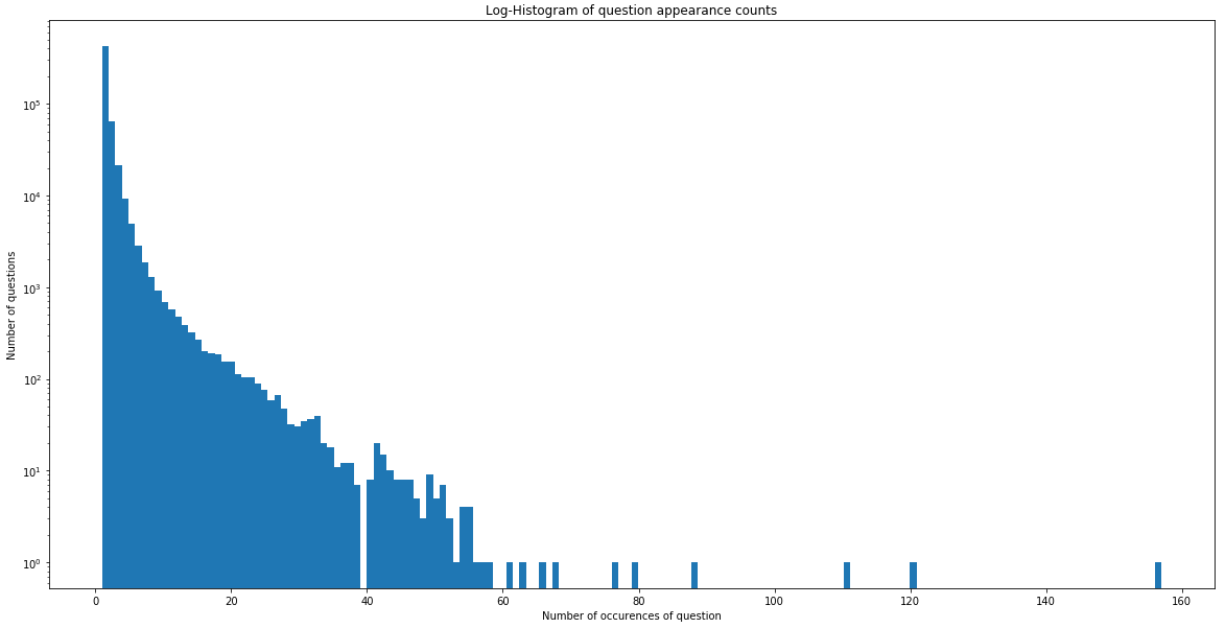
In [12]:
```
#checking whether there are any repeated pair of questions

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

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

```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [13]:
```
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'.f
```

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



3.2.5 Checking for NULL values

In [14]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

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

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

- There are two rows with null values in question2

In [15]:
```
# 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 [16]:
```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding=
```

```
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
        w2 = set(map(lambda word: word.lower().strip(), row['question2
        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
        w2 = set(map(lambda word: word.lower().strip(), row['question2
        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
        w2 = set(map(lambda word: word.lower().strip(), row['question2
        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[16]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q
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 [17]:
```
print ("Minimum length of the questions in question1 : " , min(df['q1_l

print ("Minimum length of the questions in question2 : " , min(df['q2_l

print ("Number of Questions with minimum length [question1] :", df[df[
print ("Number of Questions with minimum length [question2] :", df[df[
```

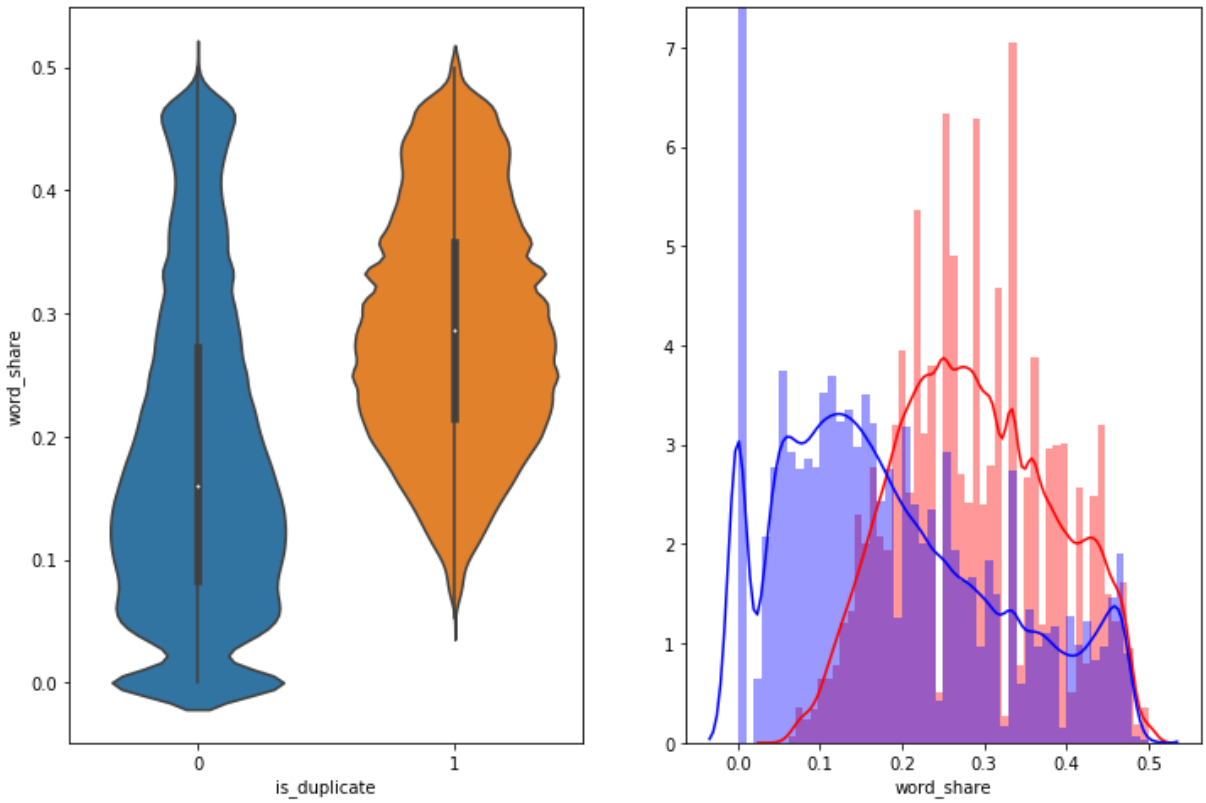
```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

3.3.1.1 Feature: word_share

In [18]:
```
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 =
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label =
plt.show()
```

```
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/scipy/stats/
stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimensional indexing is deprecat
ed; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this
will be interpreted as an array index, `arr[np.array(seq)]`, which w
ill result either in an error or a different result.
```

```
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.

/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```



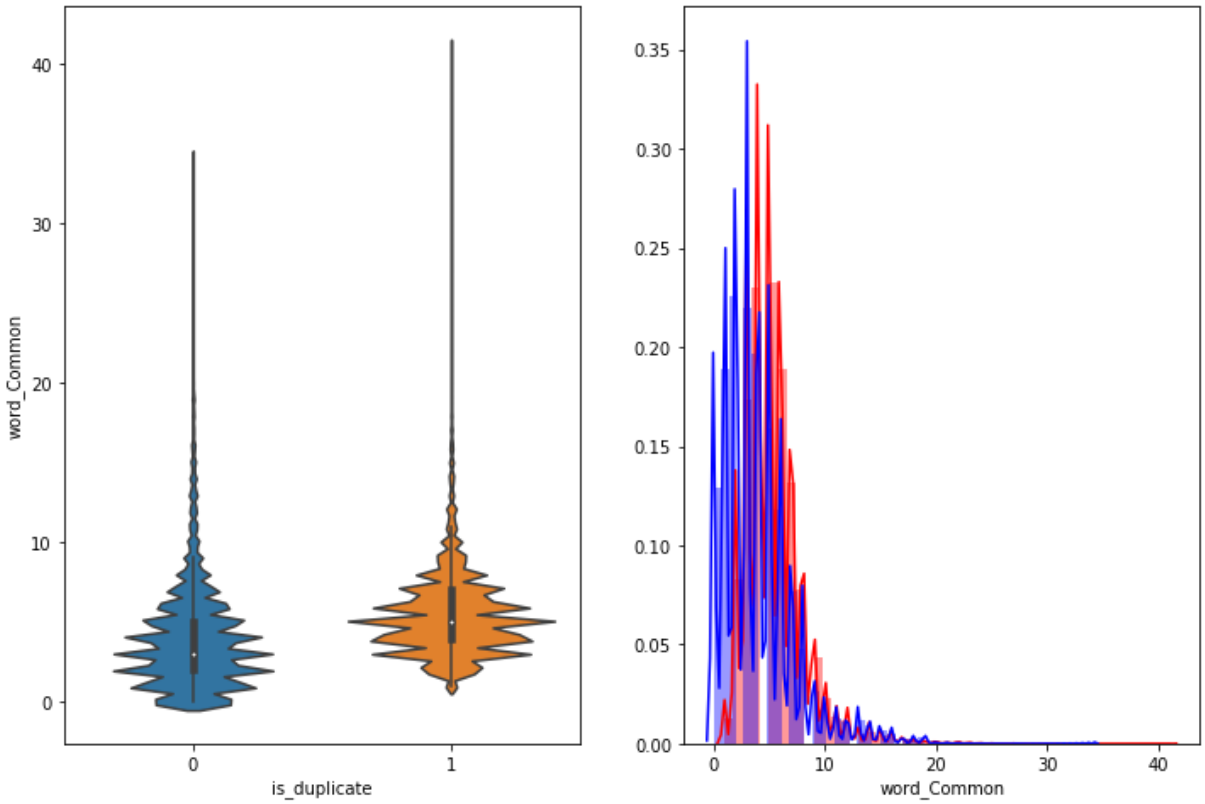
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [19]:
```
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 =
```

```
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label =
plt.show()
```

```
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/scipy/stats/
stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimensional indexing is deprecat
ed; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this
will be interpreted as an array index, `arr[np.array(seq)]`, which w
ill result either in an error or a different result.

/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/matplotlib/a
xes/_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'dens
ity' kwarg.

/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/matplotlib/a
xes/_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'dens
ity' kwarg.
```



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

1.2.1 : EDA: Advanced Feature Extraction.

In [21]:
```
import warnings
warnings.filterwarnings("ignore")
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 re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between
# you can write your own dp code for this
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-wordclou
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

In [22]:
```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding=
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run
```

In [23]:
```
df.head(2)
```

Out[23]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_
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 [24]:
```
# To get the results in 4 decemal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("’", "
                           .replace("won't", "will not").replace("cann
                           .replace("n't", " not").replace("what's", "
                           .replace("'ve", " have").replace("i'm", "i
                           .replace("he's", "he is").replace("she's",
                           .replace("%", " percent ").replace("₹", " r
                           .replace("€", " euro ").replace("'ll", " wi
    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 [25]:
```
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_WORI
    q2_words = set([word for word in q2_tokens if word not in STOP_WORI

    #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_
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_
    token_features[4] = common_token_count / (min(len(q1_tokens), len(
    token_features[5] = common_token_count / (max(len(q1_tokens), len(

    # 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[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

    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-fuzz
    # https://stackoverflow.com/questions/31806695/when-to-use-which-f
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ra
    # The token sort approach involves tokenizing the string in questi
    # then joining them back into a string We then compare the transfo
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_r
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["qu
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_subst
    return df
```

In [26]:
```
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[26]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_ma
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	0.99998
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	0.59998

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 [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},
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["questi

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

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

```
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_
print ("Total number of words in non duplicate pair questions :",len(t
```

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193130
```

Word Clouds generated from duplicate pair question's text

In [30]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopw
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
```

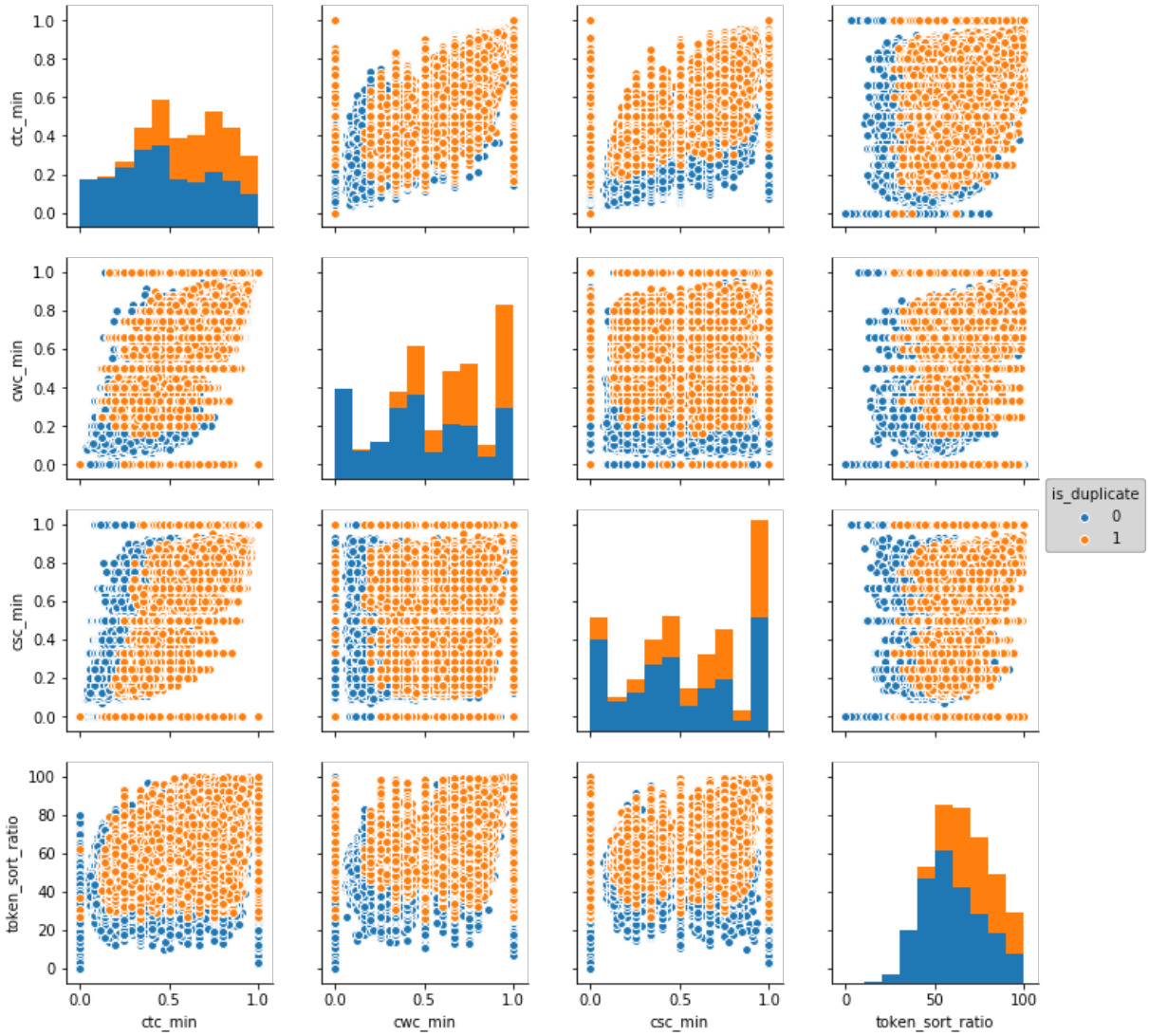
Word Clouds generated from non duplicate pair question's text

In [31]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwo
# 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()
```

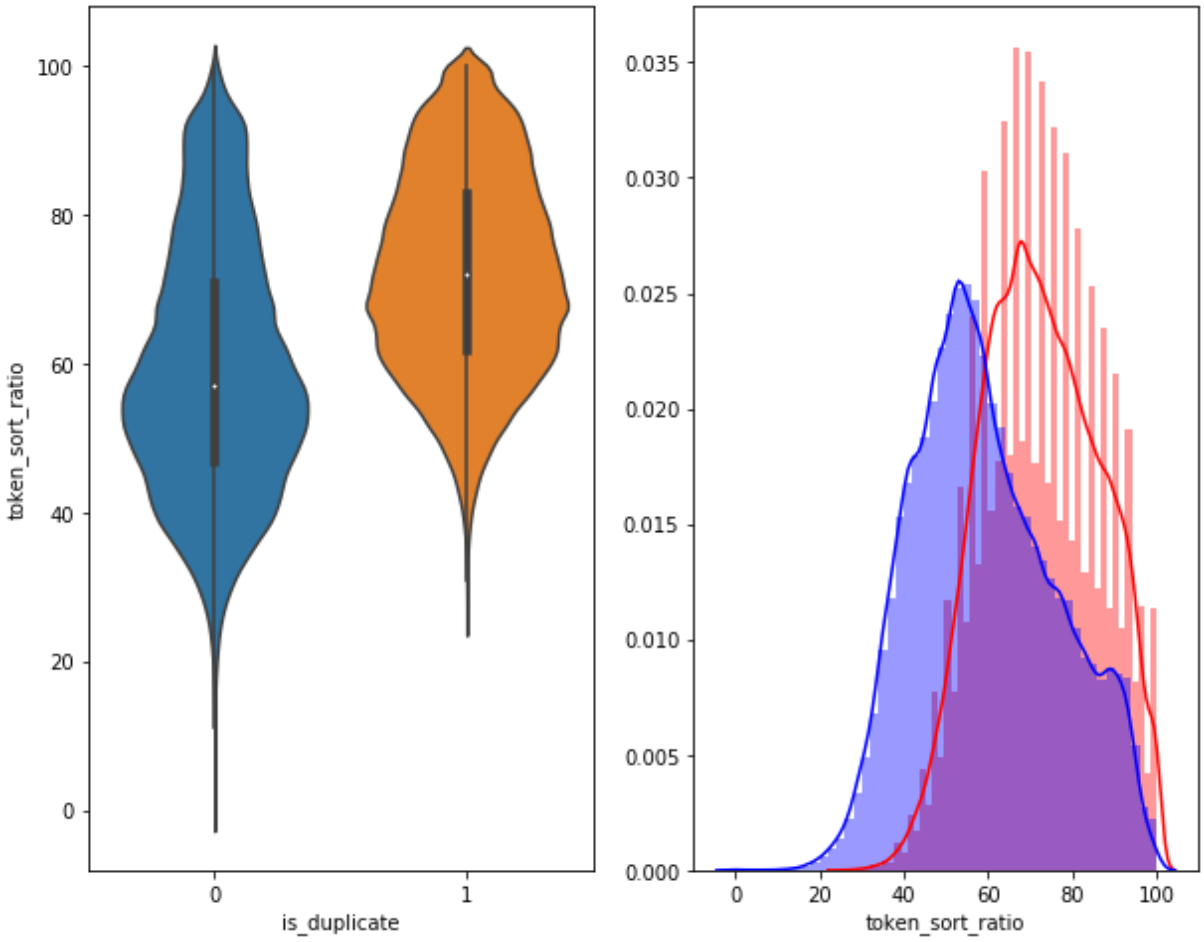
```
Word Cloud for non-Duplicate Question pairs:
```

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

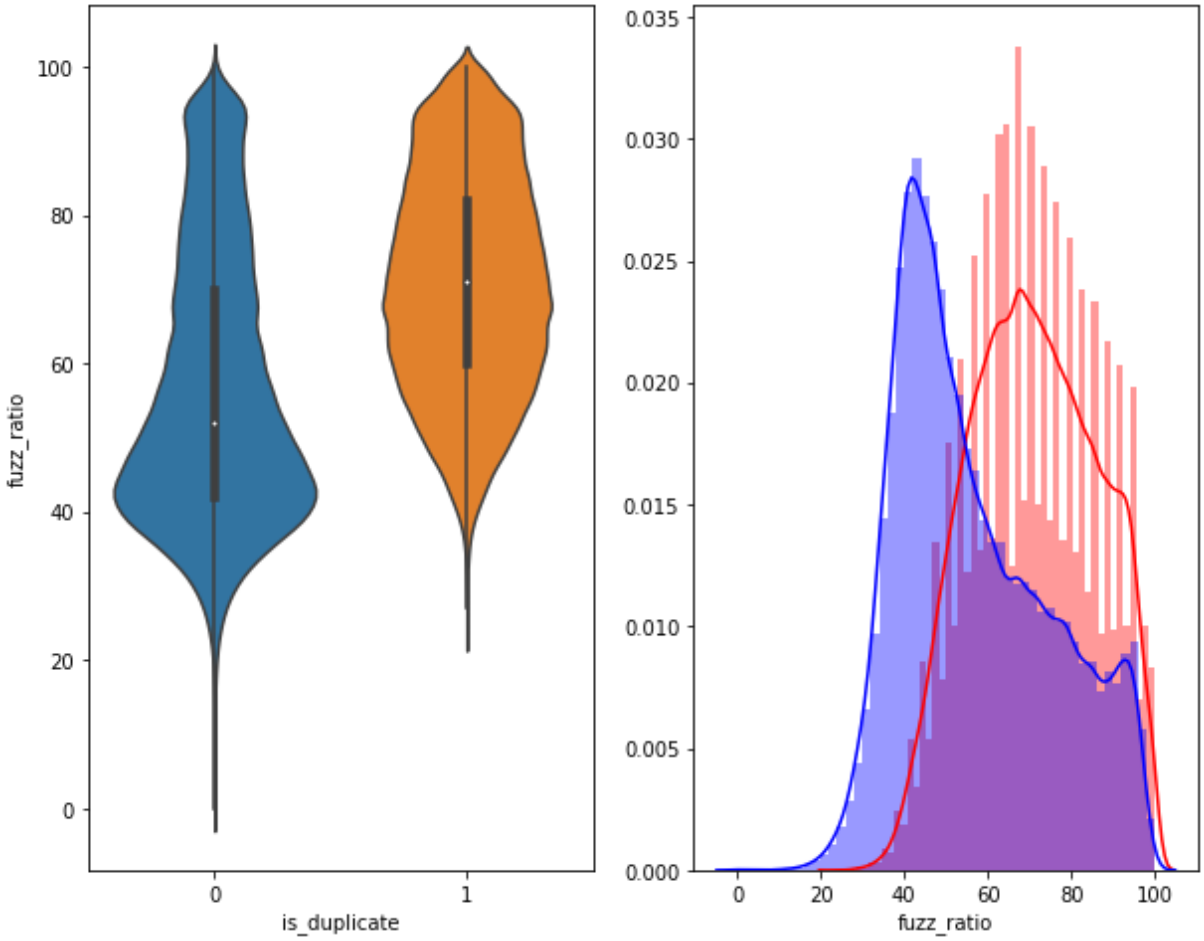
In [32]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio',
plt.show()
```



In [33]:
```
# 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
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , la
plt.show()
```



In [34]:
```
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 =
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label =
plt.show()
```

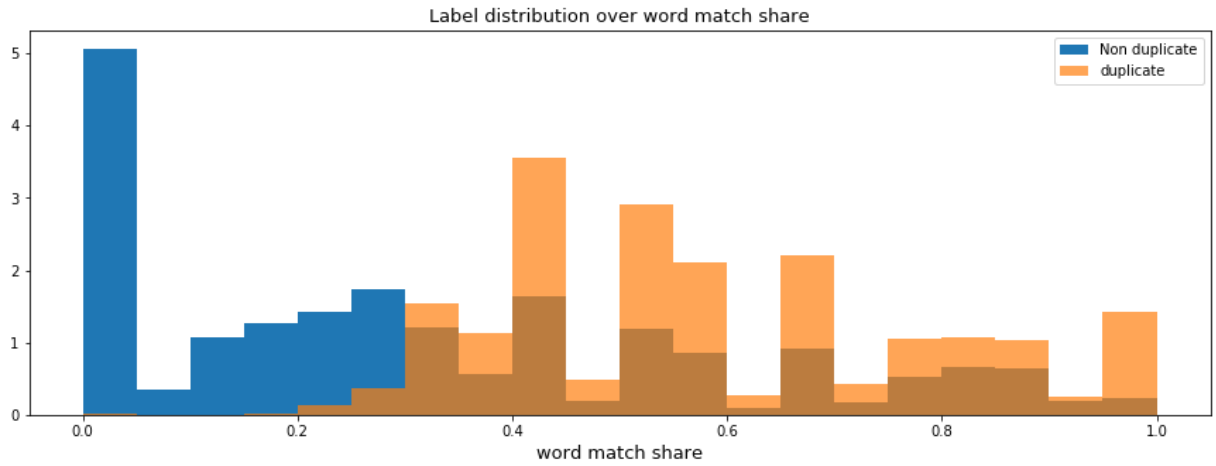


In [14]:
```
# To see the overlap of words in both Duplicate and non Duplicate data

import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')

stops = set(stopwords.words("english"))

def word_match_share(row):
    q1words = {}
    q2words = {}
    for word in str(row['question1']).lower().split():
        if word not in stops:
            q1words[word] = 1
    for word in str(row['question2']).lower().split():
```

```
    for word in str(row['question2']).lower().split():
        if word not in stops:
            q2words[word] = 1
    if len(q1words) == 0 or len(q2words) == 0:
        # The computer-generated chaff includes a few questions that a
        return 0
    shared_words_in_q1 = [w for w in q1words.keys() if w in q2words]
    shared_words_in_q2 = [w for w in q2words.keys() if w in q1words]
    R = (len(shared_words_in_q1) + len(shared_words_in_q2))/(len(q1wor
    return R

plt.figure(figsize=(15, 5))
train_word_match = df.apply(word_match_share, axis=1, raw=True)
plt.hist(train_word_match[df['is_duplicate'] == 0], bins=20, normed=Tr
plt.hist(train_word_match[df['is_duplicate'] == 1], bins=20, normed=Tr
plt.legend()
plt.title('Label distribution over word match share', fontsize=13)
plt.xlabel('word match share', fontsize=13)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/rohitbohra/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/matplotlib/a
xes/_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'dens
ity' kwarg.

/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/matplotlib/a
xes/_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'dens
ity' kwarg.
```

Out[14]: Text(0.5,0,'word match share')



3.5.2 Visualization

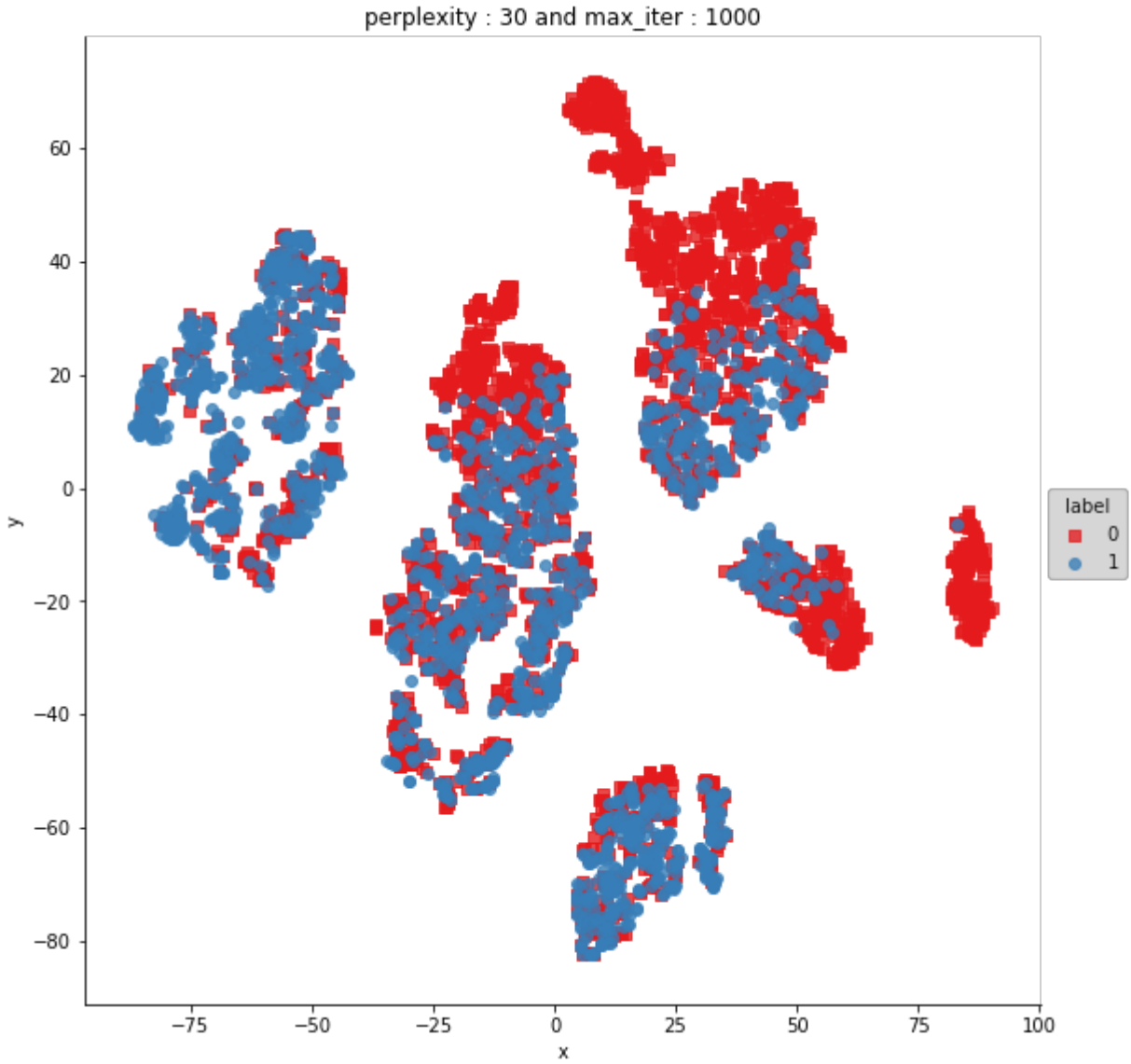
In [37]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated af
from sklearn.preprocessing import MinMaxScaler
dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max',
y = dfp_subsampled['is_duplicate'].values
```

In [38]:
```
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 0.112s...
[t-SNE] Computed neighbors for 5000 samples in 0.402s...
[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 0.338s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600
(50 iterations in 3.549s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003
(50 iterations in 2.524s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721
(50 iterations in 2.426s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246
(50 iterations in 2.529s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275
(50 iterations in 2.396s)
[t-SNE] KL divergence after 250 iterations with early exaggeration:
67.273346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933
(50 iterations in 2.498s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826
(50 iterations in 2.597s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772
(50 iterations in 2.445s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877
(50 iterations in 2.422s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429
(50 iterations in 2.555s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178
(50 iterations in 2.596s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036
(50 iterations in 2.696s)
```

```
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951
(50 iterations in 2.650s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860
(50 iterations in 2.384s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789
(50 iterations in 2.474s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745
(50 iterations in 2.539s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732
(50 iterations in 2.583s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689
(50 iterations in 2.558s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651
(50 iterations in 2.458s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590
(50 iterations in 2.528s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [39]:
```
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
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



In [40]:
```
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 0.456s...
[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 0.222s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941
(50 iterations in 10.598s)
[t-SNE] Iteration 100: error = 69.1120148, gradient norm = 0.0033901
(50 iterations in 5.978s)
[t-SNE] Iteration 150: error = 67.6176224, gradient norm = 0.0017826
(50 iterations in 5.026s)
[t-SNE] Iteration 200: error = 67.0574570, gradient norm = 0.0014586
(50 iterations in 4.982s)
[t-SNE] Iteration 250: error = 66.7299194, gradient norm = 0.0009065
(50 iterations in 5.128s)
[t-SNE] KL divergence after 250 iterations with early exaggeration:
66.729919
[t-SNE] Iteration 300: error = 1.4958616, gradient norm = 0.0006863
(50 iterations in 6.502s)
[t-SNE] Iteration 350: error = 1.1540339, gradient norm = 0.0001894
(50 iterations in 8.371s)
[t-SNE] Iteration 400: error = 1.0091627, gradient norm = 0.0000964
(50 iterations in 8.455s)
[t-SNE] Iteration 450: error = 0.9373680, gradient norm = 0.0000611
(50 iterations in 8.310s)
[t-SNE] Iteration 500: error = 0.9012471, gradient norm = 0.0000540
(50 iterations in 8.239s)
[t-SNE] Iteration 550: error = 0.8821378, gradient norm = 0.0000498
(50 iterations in 8.112s)
[t-SNE] Iteration 600: error = 0.8697239, gradient norm = 0.0000389
(50 iterations in 8.695s)
[t-SNE] Iteration 650: error = 0.8608552, gradient norm = 0.0000344
(50 iterations in 8.884s)
[t-SNE] Iteration 700: error = 0.8536769, gradient norm = 0.0000326
(50 iterations in 8.791s)
[t-SNE] Iteration 750: error = 0.8485754, gradient norm = 0.0000295
(50 iterations in 8.570s)
[t-SNE] Iteration 800: error = 0.8441855, gradient norm = 0.0000263
(50 iterations in 8.764s)
[t-SNE] Iteration 850: error = 0.8395877, gradient norm = 0.0000260
(50 iterations in 8.211s)
[t-SNE] Iteration 900: error = 0.8356333, gradient norm = 0.0000252
(50 iterations in 8.149s)
[t-SNE] Iteration 950: error = 0.8320156, gradient norm = 0.0000234
(50 iterations in 8.863s)
[t-SNE] Iteration 1000: error = 0.8287079, gradient norm = 0.0000247
(50 iterations in 8.893s)
[t-SNE] KL divergence after 1000 iterations: 0.828708
```

In [45]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
```

In [48]:
```
import spacy
```

In [49]:
```
# 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),"ut
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"ut
# ---------------- python 3 ---------------------
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [50]:
```
df.head()
```

Out[50]:

	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 [51]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])

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

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

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

In [61]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 384])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 404290/404290 [1:37:57<00:00, 69.30it/s]
```

In [62]:
```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 384])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 404290/404290 [1:47:01<00:00, 62.96it/s]
```

In [63]:
```
#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

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encod
else:
    print("download df_fe_without_preprocessing_train.csv from drive o
```

In [64]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],a
df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index
df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index
```

In [65]:
```
# dataframe of nlp features
df1.head()
```

Out[65]:

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

In [66]:
```
# data before preprocessing
df2.head()
```

Out[66]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Tota
0	0	1	1	66	57	14	12	10.0	23.
1	1	4	1	51	88	8	13	4.0	20.
2	2	1	1	73	59	14	10	4.0	24.
3	3	1	1	50	65	11	9	0.0	19.
4	4	3	1	76	39	13	7	2.0	20.

In [67]:
```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[67]:

	0	1	2	3	4	5	6	
0	121.929923	100.083890	72.497910	115.641794	-48.370878	34.619042	-172.057801	-92.
1	-78.070939	54.843740	82.738450	98.191858	-51.234827	55.013527	-39.140728	-82.
2	-5.355035	73.671816	14.376391	104.130220	1.433505	35.229108	-148.519409	-97.
3	5.778343	-34.712037	48.999637	59.699211	40.661261	-41.658728	-36.808583	24.
4	51.138184	38.587246	123.639495	53.333044	-47.062766	37.356215	-298.722758	-106.

5 rows × 384 columns

In [68]:
```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[68]:

	0	1	2	3	4	5	6	
0	125.983289	95.636493	42.114735	95.449986	-37.386302	39.400061	-148.116062	-87.8
1	-106.871908	80.290392	79.066289	59.302066	-42.175367	117.616711	-144.364265	-127.1
2	7.072889	15.513369	1.846888	85.937600	-33.808808	94.702299	-122.256845	-114.0
3	39.421528	44.136998	-24.010913	85.265864	-0.339021	-9.323150	-60.499635	-37.0
4	31.950112	62.854124	1.778160	36.218744	-45.130865	66.674886	-106.342343	-22.9

5 rows × 384 columns

In [69]:
```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape
print("Number of features in question2 w2v  dataframe :", df3_q2.shape
print("Number of features in final dataframe  :", df1.shape[1]+df2.sha
```

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

In [70]:
```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1['id']=df1['id']
    df3_q2['id']=df1['id']
    df1  = df1.merge(df2, on='id',how='left')
    df2  = df3_q1.merge(df3_q2, on='id',how='left')
    result  = df1.merge(df2, on='id',how='left')
    result.to_csv('final_features.csv')
```

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [2]:
```
#Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('final_features.csv', names=['Unnamed: 0','i
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
```

In [3]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None


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

In [4]:
```
read_db = 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

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

In [5]:
```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;

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

In [6]:
```
# remove the first row
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace
```

In [7]:
```
data.head()
```

Out[7]:

	cwc_min	cwc_max	csc_min	csc_max	
1	0.599988000239995	0.29999700003	0.0	0.0	0.27272479
2	0.999975000624984	0.799984000319994	0.66664444518516	0.66664444518516	0.85713061
3	0.999950002499875	0.999950002499875	0.999950002499875	0.66664444518516	0.99997500
4	0.66664444518516	0.499987500312492	0.833319444675922	0.833319444675922	0.77776913
5	0.749981250468738	0.749981250468738	0.999980000399992	0.999980000399992	0.88887901

5 rows × 794 columns

4.2 Converting strings to numerics

In [8]:
```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model
cols = list(data.columns)
for i in cols:
    data[i] = data[i].apply(pd.to_numeric)
    print(i)
```

```
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
```

In [9]:
```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a
y_true = list(map(int, y_true.values))
```

4.3 Random train test split(70:30)

In [10]:
```
X_train,X_test, y_train, y_test = train_test_split(data, y_true, strat
```

In [11]:
```
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 : (7000, 794)
Number of data points in test data : (3000, 794)
```

In [12]:
```
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_
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_di
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6292857142857143 Class 1:  0.3707142857142857
---------- Distribution of output variable in train data ----------
Class 0:  0.37066666666666664 Class 1:  0.37066666666666664
```

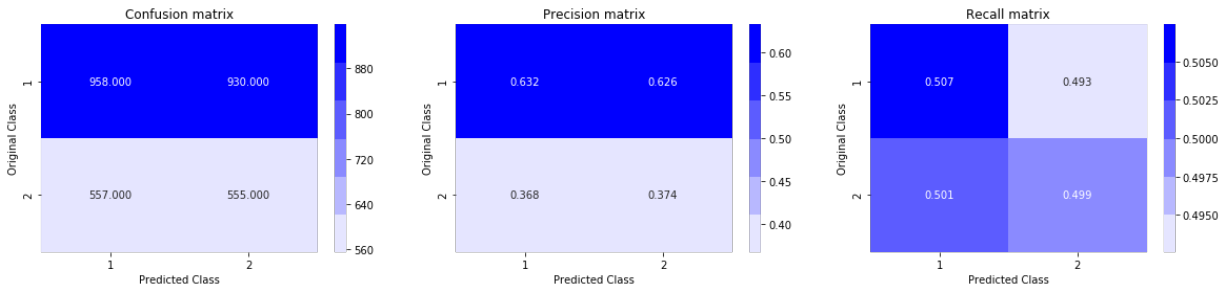
In [1]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of c

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elemen

    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresp
    # 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 elemen
    # C = [[1, 2],
    #      [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresp
    # 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=label
    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
    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
    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 [14]:
```
# 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
# 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, pred

predicted_y =np.argmax(predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

```
Log loss on Test Data using Random Model 0.8810806408444161
```



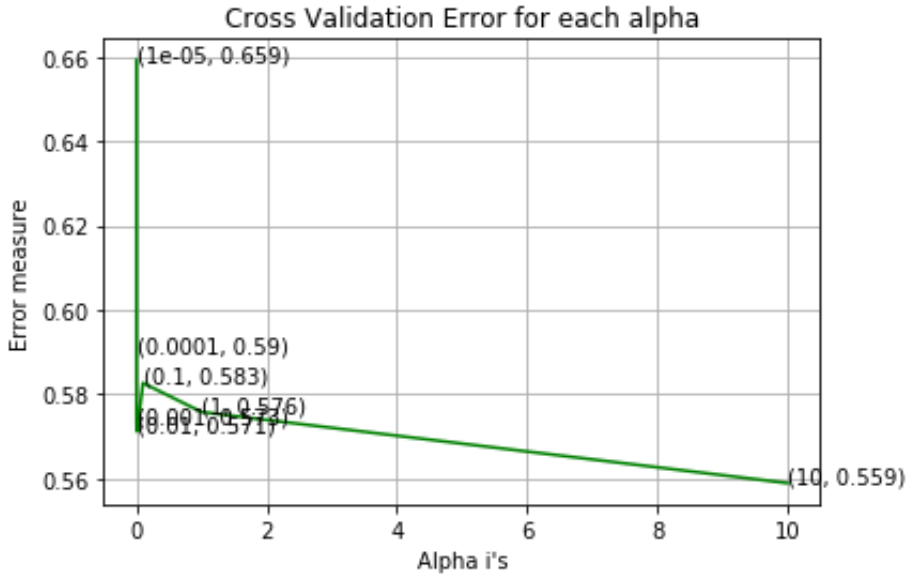
4.4 Logistic Regression with hyperparameter tuning

In [15]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie

# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
```

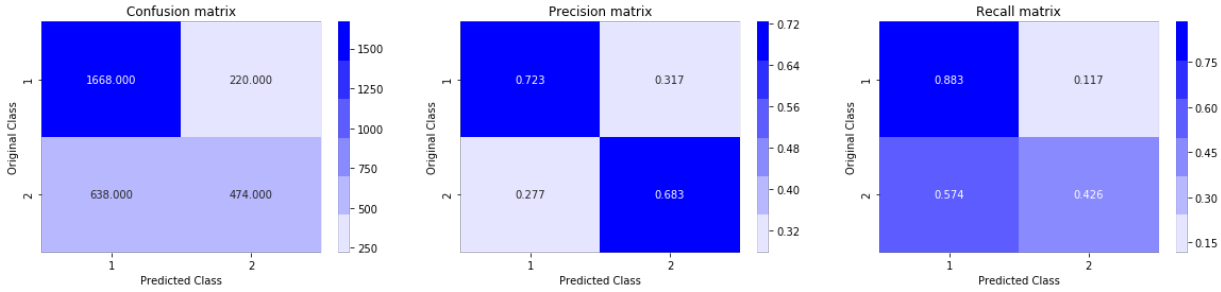
```
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stoc
# 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_stat
    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.class
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_t

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[
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',
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
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log
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: 0.6593095925368193
For values of alpha =  0.0001 The log loss is: 0.5898309216113284
For values of alpha =  0.001 The log loss is: 0.5730710478191248
For values of alpha =  0.01 The log loss is: 0.5711177491144349
```

```
For values of alpha =  0.1 The log loss is: 0.5826382802051046
For values of alpha =  1 The log loss is: 0.5757433168365903
For values of alpha =  10 The log loss is: 0.5588714916655657
```




```
For values of best alpha =  10 The train log loss is: 0.533059984692
7389
For values of best alpha =  10 The test log loss is: 0.5588714916655
657
Total number of data points : 3000
```

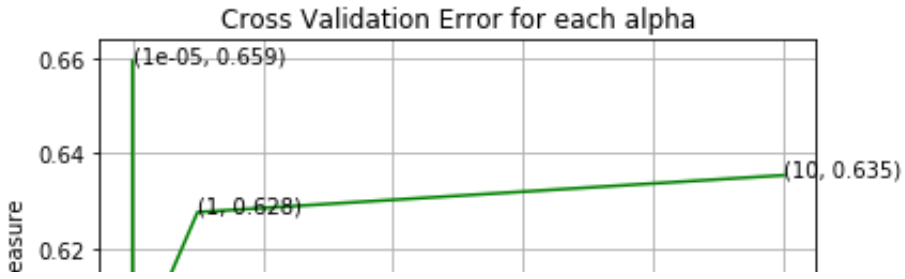


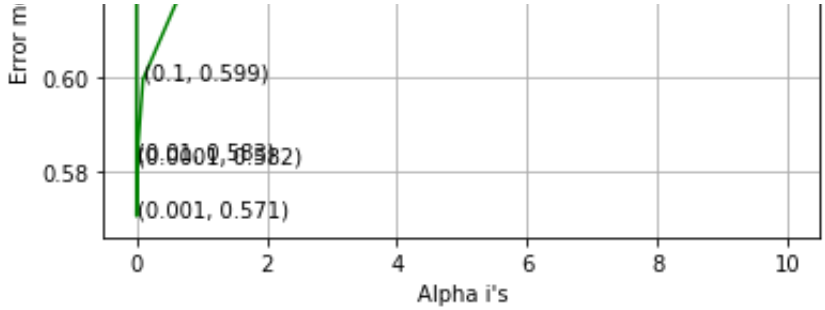
4.6 Linear SVM with hyperparameter tuning

In [16]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie

# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# predict(X)    Predict class labels for samples in X.

#------------------------------
```

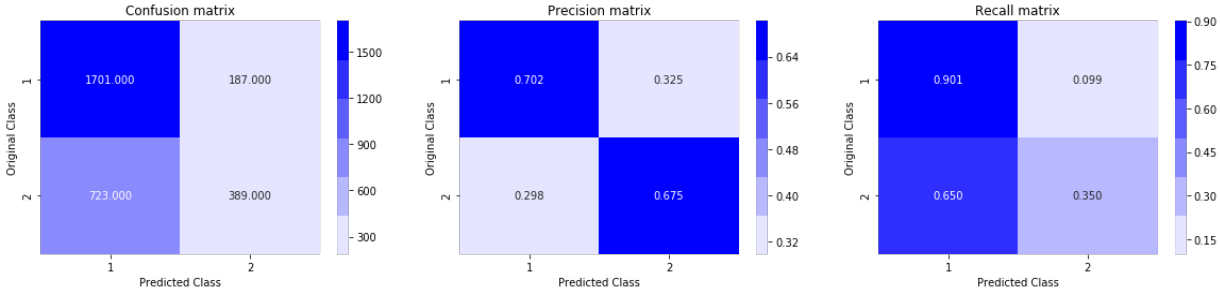
```
# video link:
#------------------------------


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_st
    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.class
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_t

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[
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
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
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log
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: 0.6593095925368193
For values of alpha =  0.0001 The log loss is: 0.5821297573176588
For values of alpha =  0.001 The log loss is: 0.570839851213848
For values of alpha =  0.01 The log loss is: 0.5828462179019105
For values of alpha =  0.1 The log loss is: 0.5994062289297418
For values of alpha =  1 The log loss is: 0.6277042959630756
For values of alpha =  10 The log loss is: 0.6354154541049325
```






```
For values of best alpha =  0.001 The train log loss is: 0.540059740
2425588
For values of best alpha =  0.001 The test log loss is: 0.5708398512
13848
Total number of data points : 3000
```



4.7 XGBoost

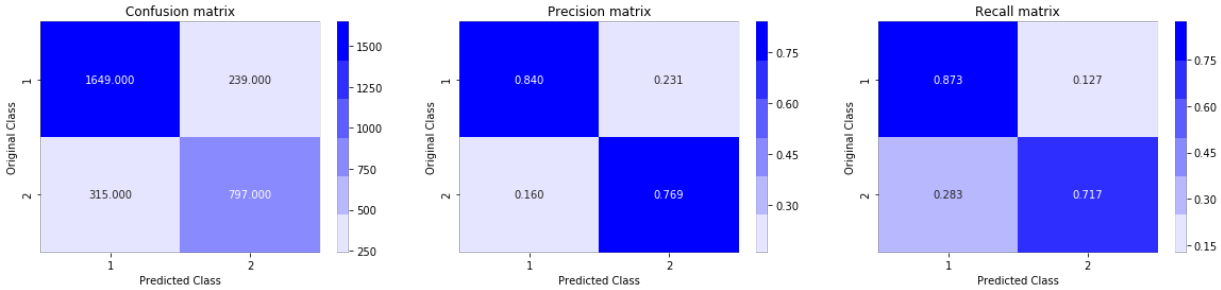
In [20]:
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

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=

xgdmat = xgb.DMatrix(X_train,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.c
```

```
[0]     train-logloss:0.684957  valid-logloss:0.684995
Multiple eval metrics have been passed: 'valid-logloss' will be used
for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.614127  valid-logloss:0.616035
[20]    train-logloss:0.562572  valid-logloss:0.56637
```

```
[30]    train-logloss:0.523381  valid-logloss:0.529237
[40]    train-logloss:0.493051  valid-logloss:0.500407
[50]    train-logloss:0.46907   valid-logloss:0.477959
[60]    train-logloss:0.449738  valid-logloss:0.460013
[70]    train-logloss:0.434002  valid-logloss:0.446085
[80]    train-logloss:0.420693  valid-logloss:0.434406
[90]    train-logloss:0.409686  valid-logloss:0.425091
[100]   train-logloss:0.400386  valid-logloss:0.417794
[110]   train-logloss:0.392153  valid-logloss:0.411717
[120]   train-logloss:0.385242  valid-logloss:0.40685
[130]   train-logloss:0.378934  valid-logloss:0.402604
[140]   train-logloss:0.373539  valid-logloss:0.399153
[150]   train-logloss:0.368156  valid-logloss:0.395822
[160]   train-logloss:0.363277  valid-logloss:0.393109
[170]   train-logloss:0.358426  valid-logloss:0.390569
[180]   train-logloss:0.354422  valid-logloss:0.388528
[190]   train-logloss:0.350242  valid-logloss:0.386246
[200]   train-logloss:0.346301  valid-logloss:0.384161
[210]   train-logloss:0.342129  valid-logloss:0.38246
[220]   train-logloss:0.338279  valid-logloss:0.381538
[230]   train-logloss:0.33465   valid-logloss:0.380345
[240]   train-logloss:0.330691  valid-logloss:0.379234
[250]   train-logloss:0.326504  valid-logloss:0.378414
[260]   train-logloss:0.322744  valid-logloss:0.377363
[270]   train-logloss:0.318881  valid-logloss:0.37642
[280]   train-logloss:0.315352  valid-logloss:0.375539
[290]   train-logloss:0.312288  valid-logloss:0.375356
[300]   train-logloss:0.308626  valid-logloss:0.374733
[310]   train-logloss:0.30532   valid-logloss:0.374409
[320]   train-logloss:0.302134  valid-logloss:0.373561
[330]   train-logloss:0.299097  valid-logloss:0.373299
[340]   train-logloss:0.295915  valid-logloss:0.372812
[350]   train-logloss:0.292799  valid-logloss:0.372506
[360]   train-logloss:0.289745  valid-logloss:0.372259
[370]   train-logloss:0.287144  valid-logloss:0.3717
[380]   train-logloss:0.284075  valid-logloss:0.371506
[390]   train-logloss:0.28131   valid-logloss:0.371238
[399]   train-logloss:0.278953  valid-logloss:0.370834
The test log loss is: 0.3708338715378971
```

In [21]:
```
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 : 3000
```



USING ONLY TF-IDF Featurization instead of TF-IDF Weighted W2V technique

In [178]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
import spacy
# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
```

In [179]:
```
# Load Basic Features
df_basic_feature = pd.read_csv("df_fe_without_preprocessing_train.csv"
```

In [180]:
```
#Number of columns in dataframe
len(df_basic_feature.columns)
```

Out[180]: 17

In [181]:
```
# list of names of columns
list(df_basic_feature.columns.values)
```

Out[181]:
```
['id',
 'qid1',
 'qid2',
 'question1',
 'question2',
 'is_duplicate',
 'freq_qid1',
 'freq_qid2',
 'q1len',
 'q2len',
 'q1_n_words',
 'q2_n_words',
 'word_Common',
 'word_Total',
 'word_share',
 'freq_q1+q2',
 'freq_q1-q2']
```

In [182]:
```
# Loading the advanced features
df_advance_features = pd.read_csv("nlp_features_train.csv",encoding='l
```

In [183]:
```
#Number of columns in dataframe
len(df_advance_features.columns)
```

Out[183]:
```
21
```

In [184]:
```
# list of names of columns
list(df_advance_features.columns.values)
```

Out[184]:
```
['id',
 'qid1',
 'qid2',
 'question1',
 'question2',
 'is_duplicate',
 'cwc_min',
 'cwc_max',
 'csc_min',
 'csc_max',
 'ctc_min',
 'ctc_max',
 'last_word_eq',
 'first_word_eq',
 'abs_len_diff',
 'mean_len',
 'token_set_ratio',
 'token_sort_ratio',
 'fuzz_ratio',
 'fuzz_partial_ratio',
 'longest_substr_ratio']
```

In [185]:
```
# Columns dropped from basic feature dataframe
df_basic_feature = df_basic_feature.drop(['qid1','qid2'],axis=1)

# Columns dropped from advance feature dataframe
df_advance_features = df_advance_features.drop(['qid1','qid2','questio

# Lets add both the truncated dataframe into one dataframe
df_basic_advance_features  = df_basic_feature.merge(df_advance_feature
```

In [186]:
```
list(df_basic_advance_features.columns.values)
```

Out[186]:
```
['id',
 'question1',
 'question2',
 'is_duplicate',
 'freq_qid1',
 'freq_qid2',
 'q1len',
 'q2len',
 'q1_n_words',
 'q2_n_words',
 'word_Common',
 'word_Total',
 'word_share',
 'freq_q1+q2',
 'freq_q1-q2',
 'cwc_min',
 'cwc_max',
 'csc_min',
 'csc_max',
 'ctc_min',
 'ctc_max',
 'last_word_eq',
 'first_word_eq',
 'abs_len_diff',
 'mean_len',
 'token_set_ratio',
 'token_sort_ratio',
 'fuzz_ratio',
 'fuzz_partial_ratio',
 'longest_substr_ratio']
```

In [187]:
```
y_true = df_basic_advance_features['is_duplicate']
```

In [188]:
```
df_basic_advance_features = df_basic_advance_features.drop(['id'],axis=
df_basic_advance_features = df_basic_advance_features.drop(['is_duplic
```

In [189]:
```
null_columns=df_basic_advance_features.columns[df_basic_advance_featur
df_basic_advance_features[null_columns].isnull().sum()
```

Out[189]:
```
question1    1
question2    2
dtype: int64
```

In [190]:
```
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup


# To get the results in 4 decemal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("’", "
                           .replace("won't", "will not").replace("canno
                           .replace("n't", " not").replace("what's", "w
                           .replace("'ve", " have").replace("i'm", "i a
                           .replace("he's", "he is").replace("she's",
                           .replace("%", " percent ").replace("₹", " ru
                           .replace("€", " euro ").replace("'ll", " wil
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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


    return x
```

In [255]:
```
# preprocessing each question
df_basic_advance_features['question1'] = df_basic_advance_features['que
df_basic_advance_features['question2'] = df_basic_advance_features['que
```

In [256]:
```
df_basic_advance_features['question1'] = df_basic_advance_features['que
df_basic_advance_features['question2'] = df_basic_advance_features['que
```

In [198]:
```
df_basic_advance_features["question1"] = df_basic_advance_features["que
df_basic_advance_features["question2"] = df_basic_advance_features["que
```

In [257]:
```
df_basic_advance_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404290 entries, 0 to 404289
Data columns (total 28 columns):
question1               404290 non-null object
question2               404290 non-null object
freq_qid1               404290 non-null int64
freq_qid2               404290 non-null int64
q1len                   404290 non-null int64
q2len                   404290 non-null int64
q1_n_words              404290 non-null int64
q2_n_words              404290 non-null int64
word_Common             404290 non-null float64
word_Total              404290 non-null float64
word_share              404290 non-null float64
freq_q1+q2              404290 non-null int64
freq_q1-q2              404290 non-null int64
cwc_min                 404290 non-null float64
cwc_max                 404290 non-null float64
csc_min                 404290 non-null float64
csc_max                 404290 non-null float64
ctc_min                 404290 non-null float64
ctc_max                 404290 non-null float64
last_word_eq            404290 non-null float64
first_word_eq           404290 non-null float64
abs_len_diff            404290 non-null float64
mean_len                404290 non-null float64
token_set_ratio         404290 non-null int64
token_sort_ratio        404290 non-null int64
fuzz_ratio              404290 non-null int64
fuzz_partial_ratio      404290 non-null int64
longest_substr_ratio    404290 non-null float64
dtypes: float64(14), int64(12), object(2)
memory usage: 89.5+ MB
```

In [271]:
```
x_train,x_test, y_train, y_test = train_test_split(df_basic_advance_fe
```

In [272]:
```
print("The shape of train data is ",x_train.shape)
print("The shape of Y train data is ",y_train.shape)
print("The shape of test data is ",x_test.shape)
print("The shape of Y test data is ",y_test.shape)
```

```
The shape of train data is  (283003, 28)
The shape of Y train data is  (283003,)
The shape of test data is  (121287, 28)
The shape of Y test data is  (121287,)
```

In [273]:
```
tfidf = TfidfVectorizer()

train_X = x_train['question1'] + x_train['question2']
question1_question2_train = tfidf.fit_transform(train_X)

test_X = x_test['question1'] + x_test['question2']
question1_question2_test = tfidf.transform(test_X)


print("The shape of test data is ",question1_question2_train.shape)
print("The shape of Y test data is ",question1_question2_test.shape)
```

```
The shape of test data is  (283003, 74077)
The shape of Y test data is  (121287, 74077)
```

In [278]:
```
x_train = x_train.drop(['question1'],axis=1)
```

In [279]:
```
x_test=x_test.drop(['question2'],axis=1)
```

In [281]:
```
x_train = x_train.drop(['question2'],axis=1)
x_test=x_test.drop(['question1'],axis=1)
```

In [282]:
```
x_train = hstack((x_train, question1_question2_train),dtype='float64')
x_test = hstack((x_test, question1_question2_test),dtype='float64').to
```

In [228]:
```
# Instanciate Tfidf Vectorizer
tfidfVectorizer_question1_train = TfidfVectorizer(ngram_range = (1,2),

question1_train = tfidfVectorizer_question1_train.fit_transform(x_trai
```

In [229]:
```
# Instanciate Tfidf Vectorizer
tfidfVectorizer_question2_train = TfidfVectorizer(ngram_range = (1,2),

question2_train = tfidfVectorizer_question2_train.fit_transform(x_trai
```

In [230]:
```
# Instanciate Tfidf Vectorizer
tfidfVectorizer_question1_test = TfidfVectorizer(ngram_range = (1,2), r

question1_test = tfidfVectorizer_question1_test.fit_transform(x_test['

tfidfVectorizer_question2_test = TfidfVectorizer(ngram_range = (1,2), r

question2_test = tfidfVectorizer_question2_test.fit_transform(x_test['
```

In [231]:
```
question1_question2_train = hstack((question1_train,question2_train))
question1_question2_test = hstack((question1_test,question2_test))
```

In [232]:
```
type(question1_question2_train)
```

Out[232]:
```
scipy.sparse.coo.coo_matrix
```

In [233]:
```
# Drop unnecessary question1 and question2 columns
x_train.drop(['question1','question2'], axis=1, inplace=True)
```

In [234]:
```
# Drop unnecessary question1 and question2 columns
x_test.drop(['question1','question2'], axis=1, inplace=True)
```

In [235]:
```
# Combine all basic, advance and tfidf features
x_train = hstack((X_train, question1_question2_train),format="csr",dty
```

In [236]:
```
# Combine all basic, advance and tfidf features
x_test = hstack((X_test, question1_question2_test),format="csr",dtype=
```

In [283]:
```
x_train.shape
```

Out[283]:
```
(283003, 74103)
```

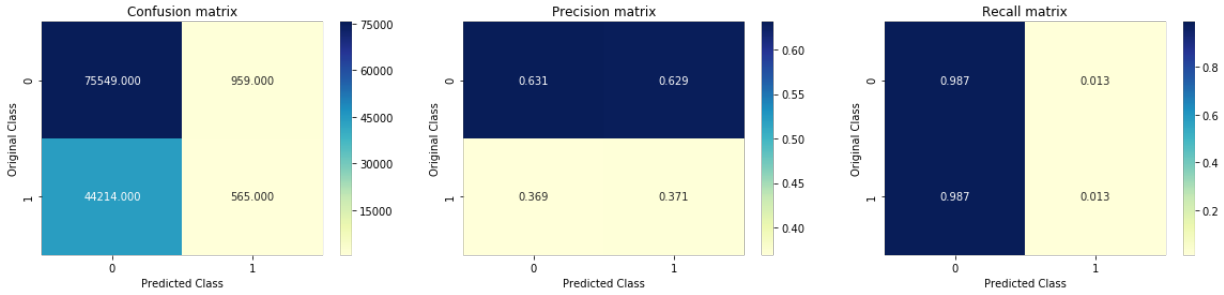
In [284]:
```
x_test.shape
```

Out[284]:
```
(121287, 74103)
```

In [285]:
```
# 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)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))

    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap='YlGnBu', fmt=".3f", xticklabels=la
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

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

    plt.show()
```

In [286]:
```
##  Models

# Random Model
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, pred

predicted_y =np.argmax(predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

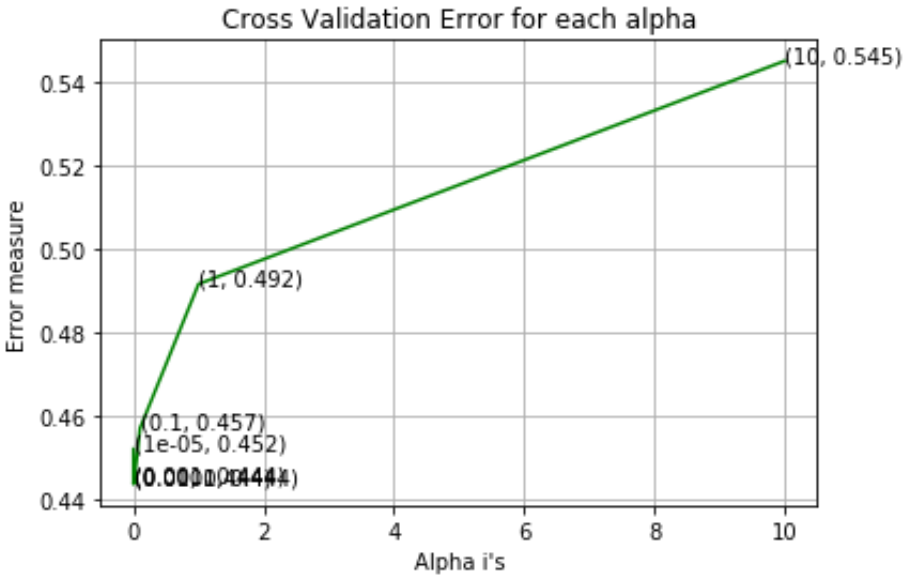
```
Log loss on Test Data using Random Model 0.6978384229962716
```



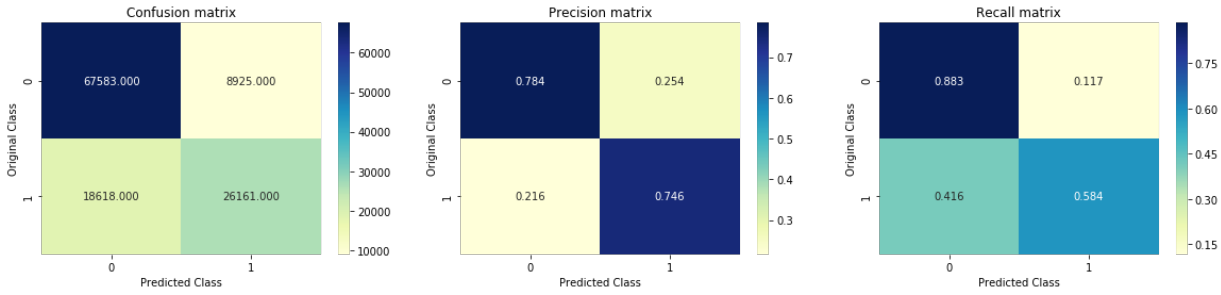
In [287]:
```
# logistic regression

alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_stat
    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.clas
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_t

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[
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',
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
predict_y = sig_clf.predict_proba(x_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log
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: 0.45222282457073765
For values of alpha =  0.0001 The log loss is: 0.4438136398168544
For values of alpha =  0.001 The log loss is: 0.44421058215133563
For values of alpha =  0.01 The log loss is: 0.44406531462283577
For values of alpha =  0.1 The log loss is: 0.45716077943043243
For values of alpha =  1 The log loss is: 0.4917796143696626
For values of alpha =  10 The log loss is: 0.5452455477724563
```



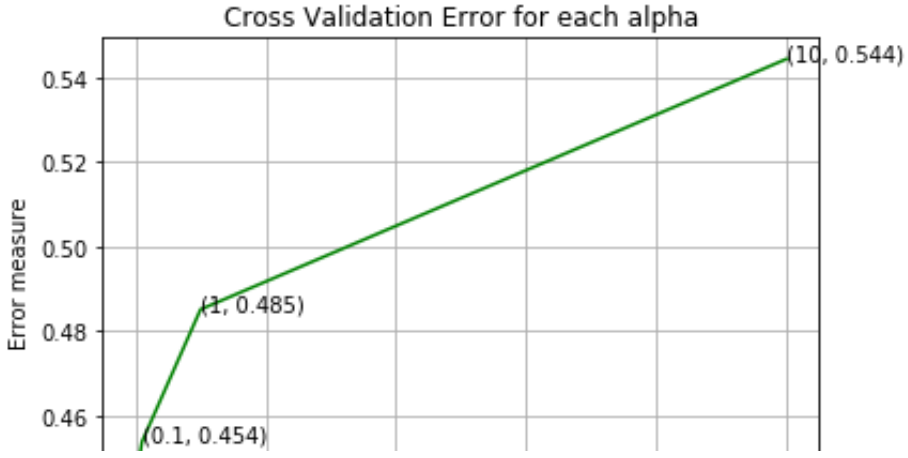

```
For values of best alpha =  0.0001 The train log loss is: 0.44614785
34986376
For values of best alpha =  0.0001 The test log loss is: 0.443813639
8168544
Total number of data points : 121287
```

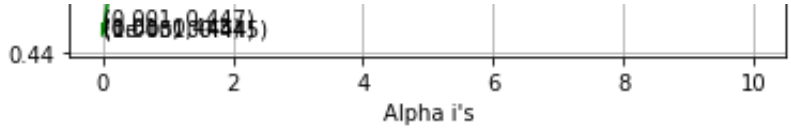


In [290]:
```
#SVM

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

log_error_array=[]
```

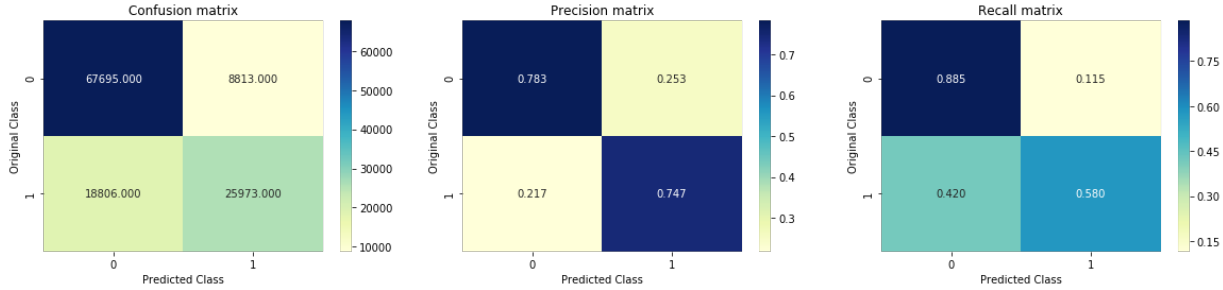
```
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_st
    clf.fit(train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train, y_train)
    predict_y = sig_clf.predict_proba(x_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.clas
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_t

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[
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='hinge
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
predict_y = sig_clf.predict_proba(x_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log
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: 0.4443182937236933
For values of alpha =  0.0001 The log loss is: 0.4446676661749702
For values of alpha =  0.001 The log loss is: 0.44695080803075365
For values of alpha =  0.01 The log loss is: 0.4449931122995938
For values of alpha =  0.1 The log loss is: 0.45382778060219536
For values of alpha =  1 The log loss is: 0.48516772493285976
For values of alpha =  10 The log loss is: 0.5444155987261744
```






```
For values of best alpha =  1e-05 The train log loss is: 0.446662500
1999097
For values of best alpha =  1e-05 The test log loss is: 0.4443182937
236933
Total number of data points : 121287
```



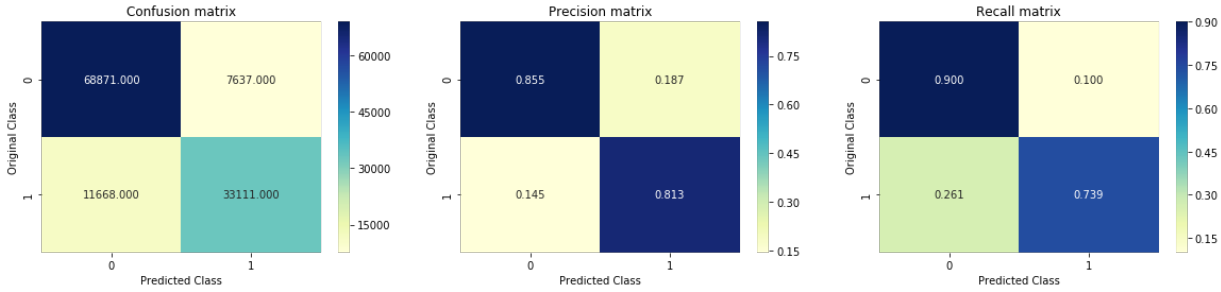
In [293]:
```
from xgboost import XGBClassifier
import scipy.stats as sc
from sklearn.model_selection import RandomizedSearchCV,StratifiedKFold
```

In [295]:
```
# Hyperparameters
learning_rate = sc.uniform(0.01,0.1)
base_learners = sc.randint(10,200)
depth = sc.randint(5,10)
min_child_weight = sc.randint(5,10)

params = {'learning_rate': learning_rate, 'n_estimators':base_learners

xgb_classifier = xgb.XGBClassifier(objective='binary:logistic')
gsv = RandomizedSearchCV(xgb_classifier, params, cv=3, scoring="neg_lo
gsv.fit(x_train,y_train)

print("Best Hyperparameter: ", gsv.best_params_)
print("Best neg_log_loss: %.2f%%",(gsv.best_score_*100))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 71.1min finish
ed

Best Hyperparameter:  {'learning_rate': 0.0364555612104627, 'max_dep
th': 8, 'min_child_weight': 7, 'n_estimators': 193}
Best neg_log_loss: %.2f%% -33.000159827153645
```

In [296]:
```
predict_y = gsv.predict_proba(x_train)
print("The train log loss is:",log_loss(y_train, predict_y, eps=1e-15))
predict_y = gsv.predict_proba(x_test)
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
The train log loss is: 0.31688220098944414
The test log loss is: 0.32862519851050037
Total number of data points : 121287
```



In [297]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ['Model Name', 'Tokenizer', 'Test Log Loss']
x.add_row(["Random model","TFIDF Weighted W2V","0.881"])
x.add_row(["Logistic Regression","TFIDF Weighted W2V","0.55887"])
x.add_row(["Linear SVM","TFIDF Weighted W2V","0.57083"])
x.add_row(["XG BOOST","TFIDF Weighted W2V","0.37083"])
x.add_row(["Random model","TFIDF","0.69783"])
x.add_row(["Logistic Regression","TFIDF","0.44381"])
x.add_row(["Linear SVM","TFIDF","0.444381"])
x.add_row(["XG BOOST","TFIDF","0.32862"])

print(x)
```

Model Name	Tokenizer	Test Log Loss
Random model	TFIDF Weighted W2V	0.881
Logistic Regression	TFIDF Weighted W2V	0.55887
Linear SVM	TFIDF Weighted W2V	0.57083
XG BOOST	TFIDF Weighted W2V	0.37083
Random model	TFIDF	0.69783
Logistic Regression	TFIDF	0.44381
Linear SVM	TFIDF	0.444381
XG BOOST	TFIDF	0.32862

Steps followed

1) FOR TF-IDF Weighted W2V Featurization

a) We build a random model, which gives a bench mark that the other models should perform better than the random model. The test log loss is 0.881

b) Model 1 is built using Logistic Regression algorithm and we got a Log loss of 0.55887.

c) Model 2 is built using Linear Regression algorithm and we got a Log loss of 0.57083.

d) Model 3 is built using XG BOOST algorithm and we got a Log loss of 0.37083.

2) For TF-IDF Featurization

a) We build a random model, which gives a bench mark that the other models should perform better than the random model. The test log loss is 0.69783

b) Model 1 is built using Logistic Regression algorithm and we got a Log loss of 0.44381.

c) Model 2 is built using Linear Regression algorithm and we got a Log loss of 0.444381.

d) Model 3 is built using XG BOOST algorithm and we got a Log loss of 0.32862.

Conclusion

1) The model was featurized using TFIDF Weighted W2V featurization and TFIDF featurization.

2) In TFIDF Weighted W2V featurization, XG BOOST algorithm performed the best with a log loss of 0.37083

3) In TFIDF featurization, XG BOOST algorithm performed the best with a log loss of 0.32862.