Quora Question Pairs Assignment

1. Business Problem

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

2.1.1 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

3. Exploratory Data Analysis

In [5]:

```
pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7
eaa1beab2b9323073815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-n
one-any.whl
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

In [4]:

```
pip install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437
aefa0d0a92ccfb38895d9416bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz (180k
B)
    100% |████████████████████████████████| 184kB 6.9MB/s
Building wheels for collected packages: distance
  Building wheel for distance (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d
0f12db1c66dbae9c5442b39b001db18e
Successfully built distance
Installing collected packages: distance
Successfully installed distance-0.1.3
```

In [18]:

```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle (2).json
```

Out[18]:

```
{'kaggle.json': b'{"username":"shamimio","key":"e57ea3117a95a3194da0e8aaa7
da68b7"}'}
```

In [0]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

#changeb the permmission
!chmod 600 ~/.kaggle/kaggle.json
```

In [19]:

```
!kaggle competitions download -c quora-question-pairs
```

```
401 - Unauthorized
```

In [8]:

```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving train.csv to train.csv
```

In [3]:

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

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
import warnings
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



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

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

3.1 Reading data and basic stats

In [4]:

```
df = pd.read_csv("train.csv")
print("the number of datapoints: ", df.shape[0])
```

```
the number of datapoints:  404290
```

In [5]:

```
df.head()
```

Out[5]:

	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 [32]:

```
df.info()
```

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

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

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

3.2.1 Distribution of data points among output classes

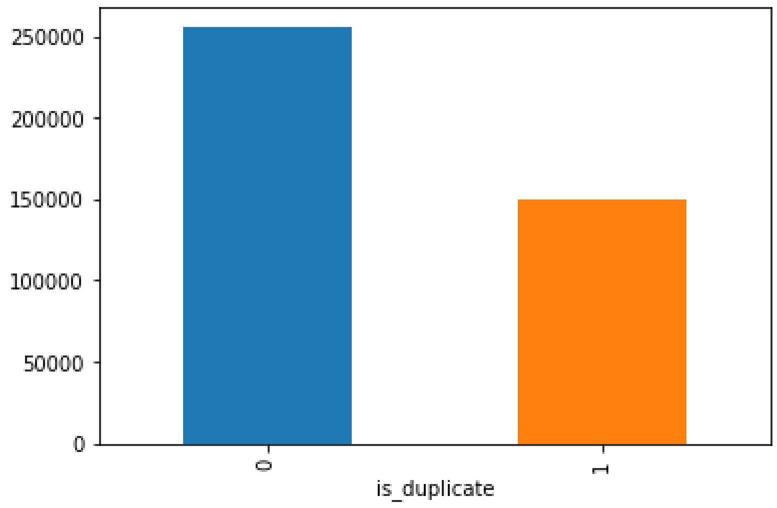
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [6]:

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

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x49a2ac77f0>
```



In [7]:

```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - roun
d(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is
_duplicate'].mean()*100, 2)))
```

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

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

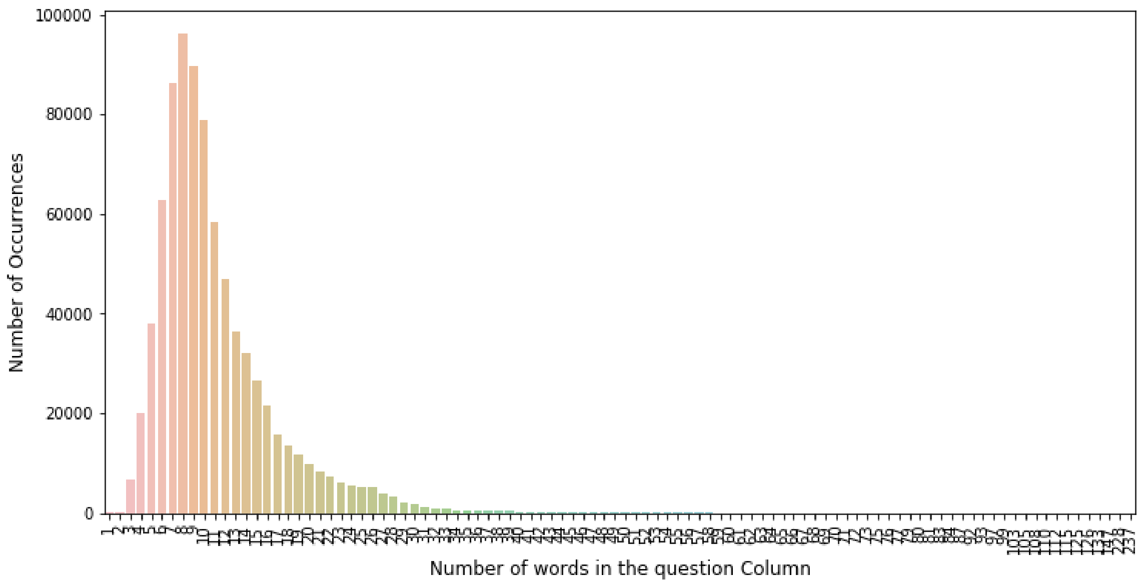
Number of words in Question column:

In [9]:

```
# appending question1 and question2
my_quest = pd.DataFrame(pd.concat([df['question1'], df['question2']]))
my_quest.columns=["questions"]

my_quest = pd.DataFrame(pd.concat([df['question1'], df['question2']]))
my_quest.columns = ["questions"]
my_quest["No_of_Words"] = my_quest["questions"].apply(lambda x : len(str(x).split()))
```

In [10]:

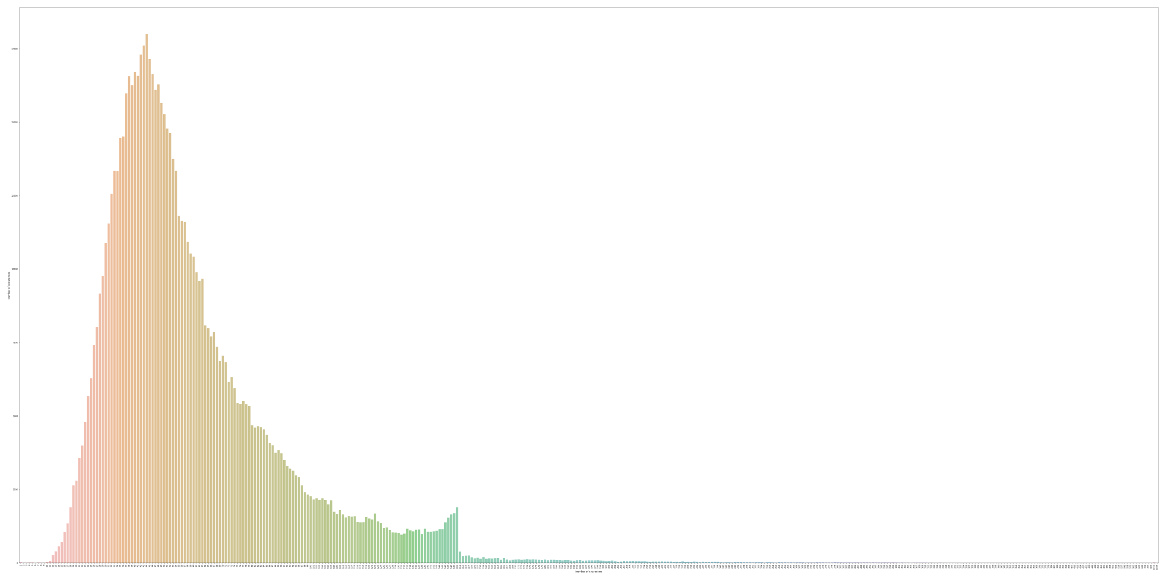
```
count =my_quest['No_of_Words'].value_counts()

plt.figure(figsize=(12,6))
sns.barplot(count.index, count.values, alpha=0.6)
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Number of words in the question Column', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```



Number of characters in the question:

In [17]:

```
my_quest["no_of_characters"]=my_quest["questions"].apply(lambda x : len(str(x)))
count = my_quest['no_of_characters'].value_counts()

plt.figure(figsize=(100,50))
sns.barplot(count.index,count.values,alpha=0.6)
plt.ylabel('Number of occurences',fontsize=12)
plt.xlabel('Number of characters',fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```



3.2.2 Number of unique questions

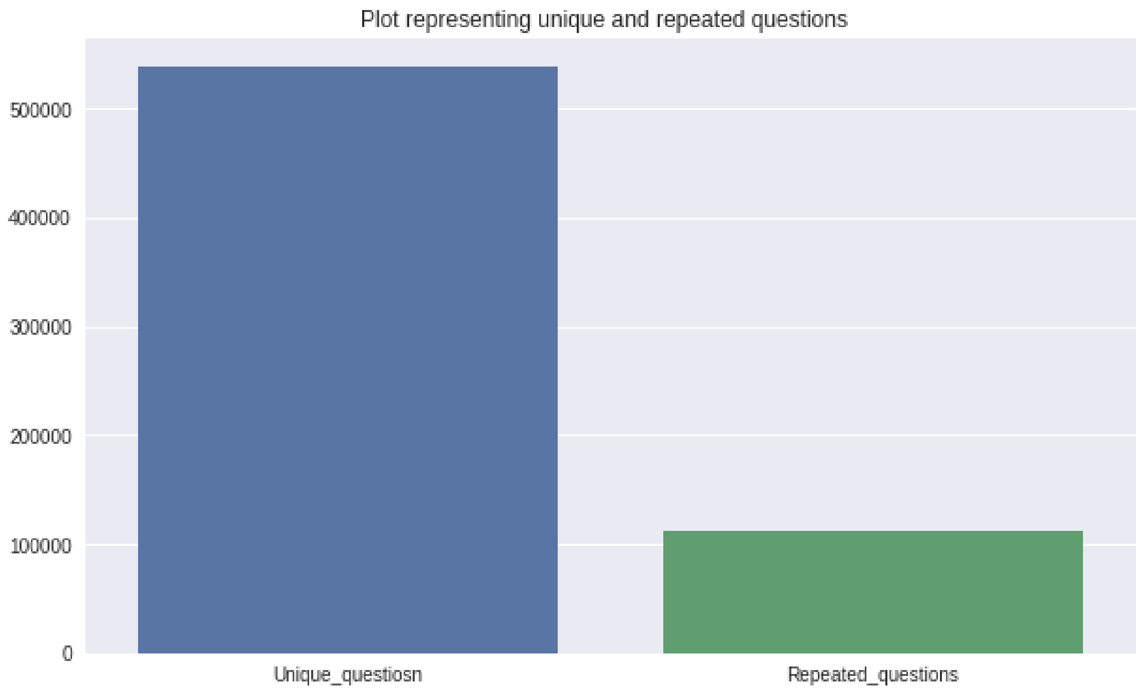
In [35]:

```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethanone = np.sum(qids.value_counts() >1 )
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(q
s_morethanone,qs_morethanone/unique_qs*100))
print ('Max number of times a single question is repeated: {}\n'.format(max(qids.value_
counts())))

q_vals=qids.value_counts()
q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.7795
3945937505%)

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

In [36]:

```
x = ["Unique_questiosn", "Repeated_questions"]
y = [unique_qs, qs_morethanone]
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 [37]:

```
#checking whether there are any repeated pair of questions

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

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

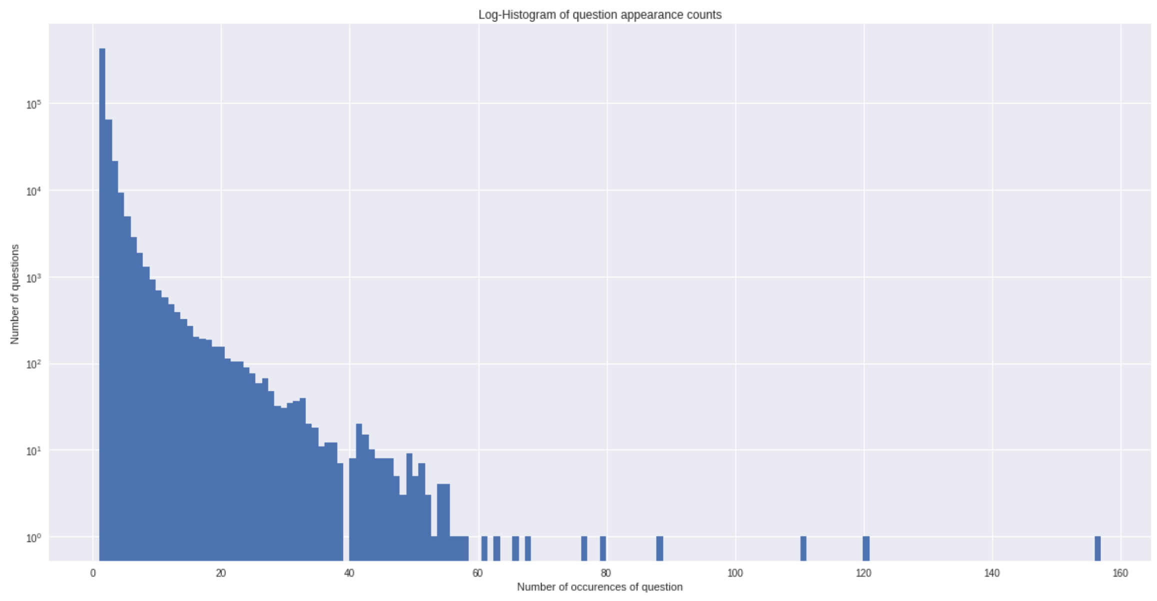
```
Number of duplicate questions 0
```

3.2.4 Number of occurrences of each question

In [38]:

```
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.va
lue_counts())))
```

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



3.2.5 Checking for NULL values

In [39]:

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

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

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

In [40]:

```
# since two values with null values
# 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 [41]:

```
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[41]:
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	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	5
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	8
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	5
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	6
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	3

3.3.1 Analysis of some of the extracted features

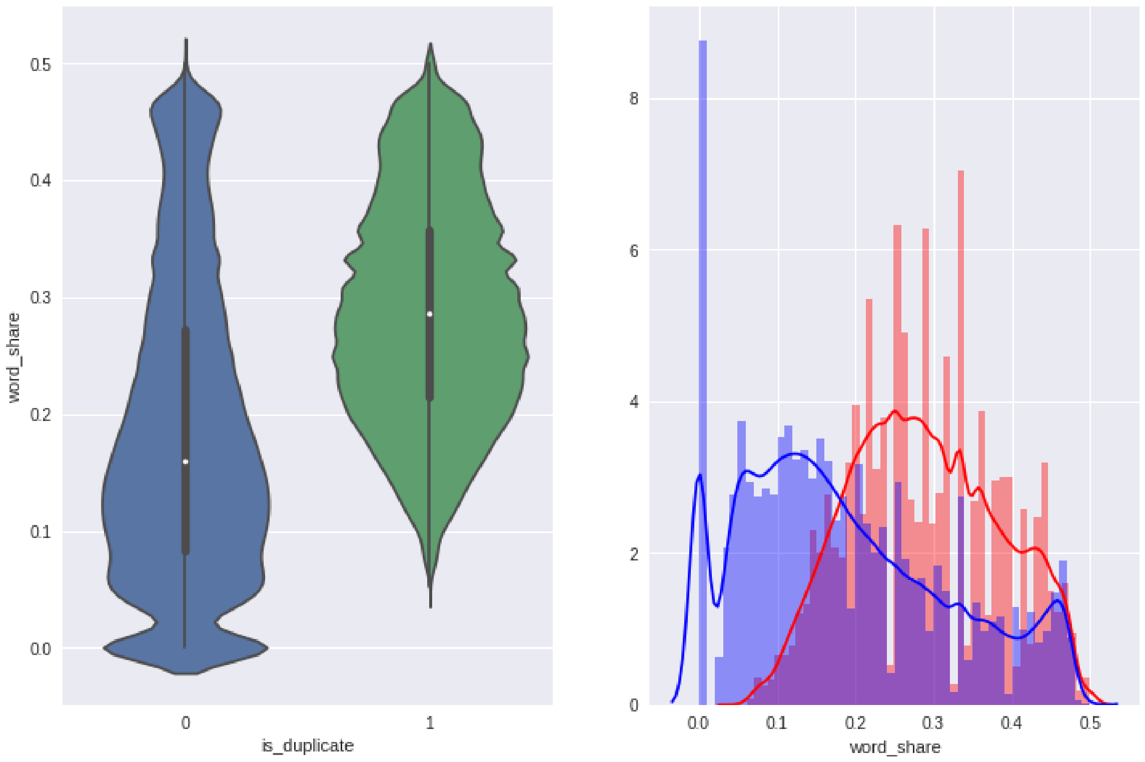
In [42]:

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

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

print ("Number of Questions with minimum length [question1] :", df[df['q1_n_words']== 1
].shape[0])
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']== 1
].shape[0])
```

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

In [43]:

```
plt.figure(figsize=(12, 8))

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

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_share'][0:] , label = "1", color = 're
d')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , color = 'b
lue' )
plt.show()
```

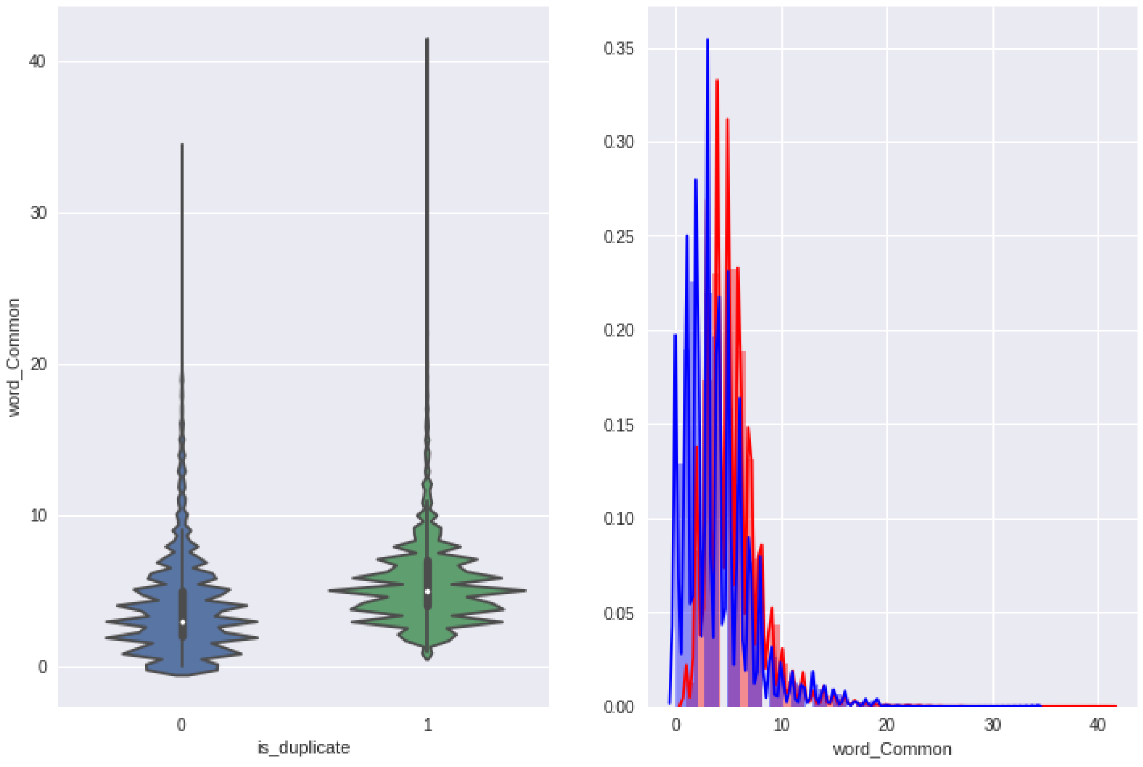


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

Feature: word_Common

In [44]:

```
plt.figure(figsize=(12, 8))

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

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



The feaures are highly overlapping.

3.4 Preprocessing of Text

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

In [45]:

```
import nltk
nltk.download('stopwords')
```

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

Out[45]:

```
True
```

In [0]:

```
# 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("’",
"'")\
                           .replace("won't", "will not").replace("cannot", "can not").r
eplace("can't", "can not")\
                           .replace("n't", " not").replace("what's", "what is").replace
("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace("'r
e", " are")\
                           .replace("he's", "he is").replace("she's", "she is").replace
("'s", " own")\
                           .replace("%", " percent ").replace("₹", " rupee ").replace(
"$", " dollar ")\
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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


    return x
```

3.5 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 [0]:

```
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_D
IV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_D
IV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_D
IV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_D
IV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAF
E_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAF
E_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["question
2"]), axis=1)

    df["cwc_min"]       = list(map(lambda x: x[0], token_features))
    df["cwc_max"]       = list(map(lambda x: x[1], token_features))
    df["csc_min"]       = list(map(lambda x: x[2], token_features))
    df["csc_max"]       = list(map(lambda x: x[3], token_features))
    df["ctc_min"]       = list(map(lambda x: x[4], token_features))
    df["ctc_max"]       = list(map(lambda x: x[5], token_features))
    df["last_word_eq"]  = list(map(lambda x: x[6], token_features))
    df["first_word_eq"] = list(map(lambda x: x[7], token_features))
    df["abs_len_diff"]  = list(map(lambda x: x[8], token_features))
    df["mean_len"]      = list(map(lambda x: x[9], token_features))

    #Computing Fuzzy Features and Merging with Dataset

    # do read this blog: http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching
-in-python/
    # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-function-to-c
ompare-2-strings
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"
], x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the t
okens alphabetically, and
    # then joining them back into a string We then compare the transformed strings with
a simple ratio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question
1"], x["question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["que
stion2"]), axis=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"],
x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["questi
on1"], x["question2"]), axis=1)
    return df
```

In [48]:

```
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)
```

```
Extracting features for train:
token features...
fuzzy features..
```

Out[48]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min
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
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

2 rows × 21 columns

3.5.1 Analysis of extracted features

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

In [49]:

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

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

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

#Saving the np array into a text file
np.savetxt('train_p.txt', p, delimiter=' ', fmt='%s')
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 [50]:

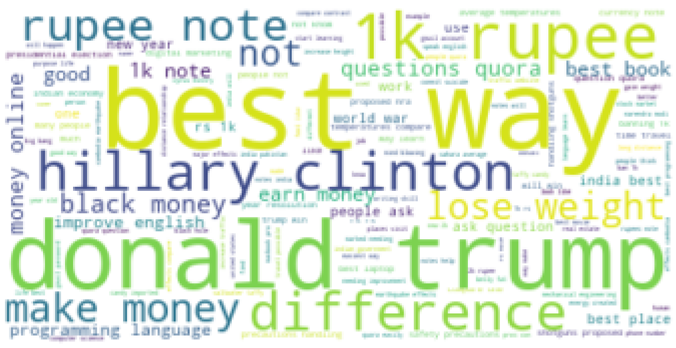
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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

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

```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193067
```

In [51]:

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

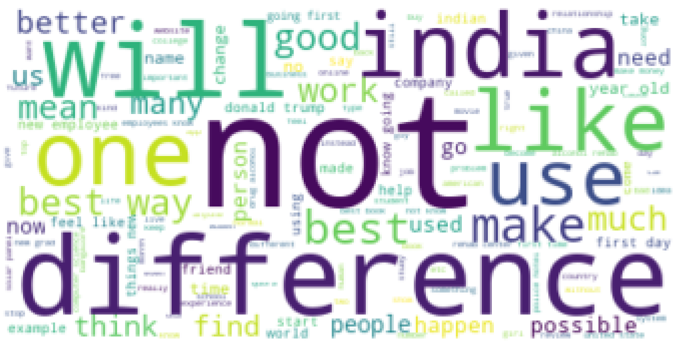
```
Word Cloud for Duplicate Question pairs
```



In [52]:

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

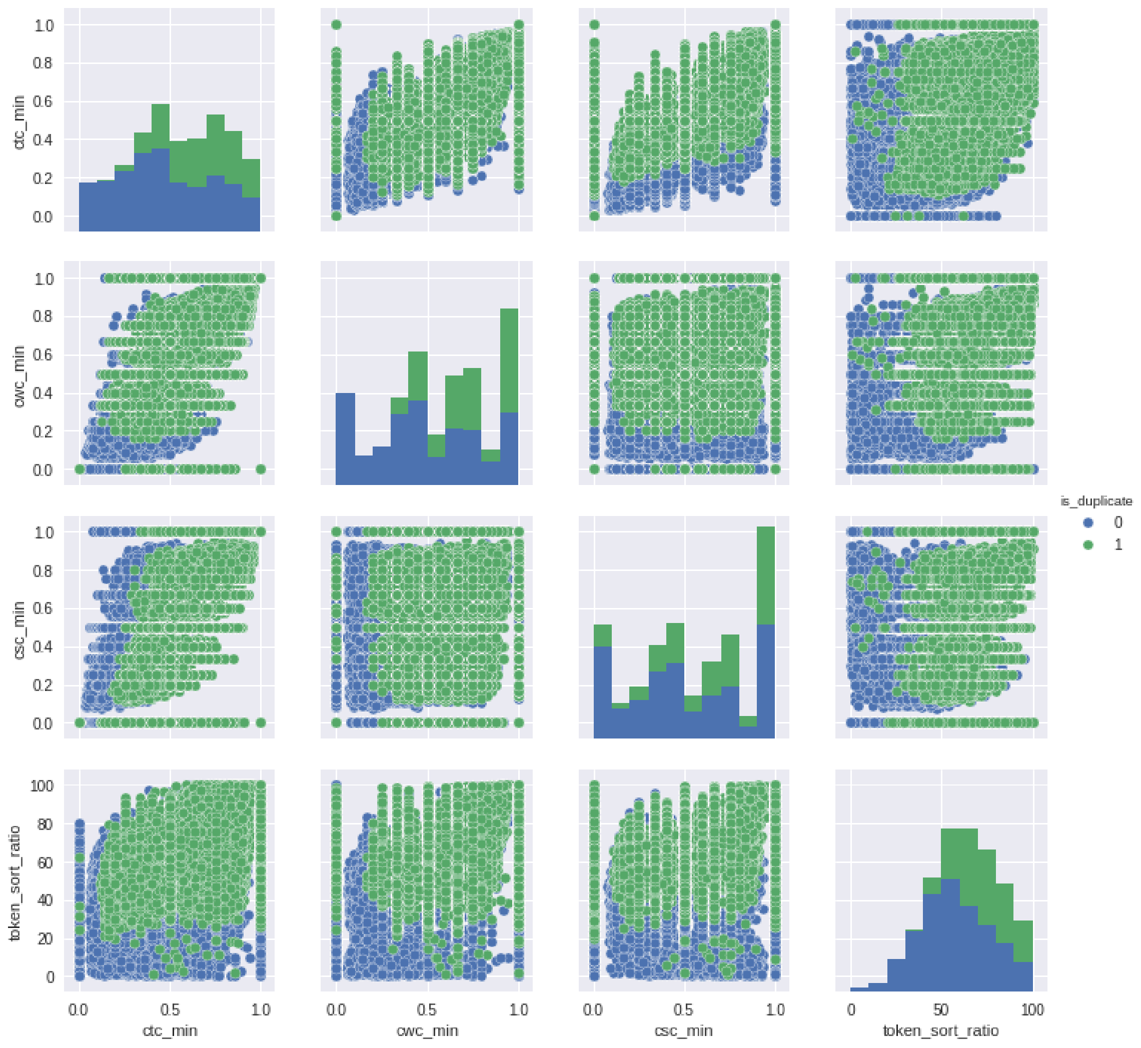
```
Word Cloud for non-Duplicate Question pairs:
```



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

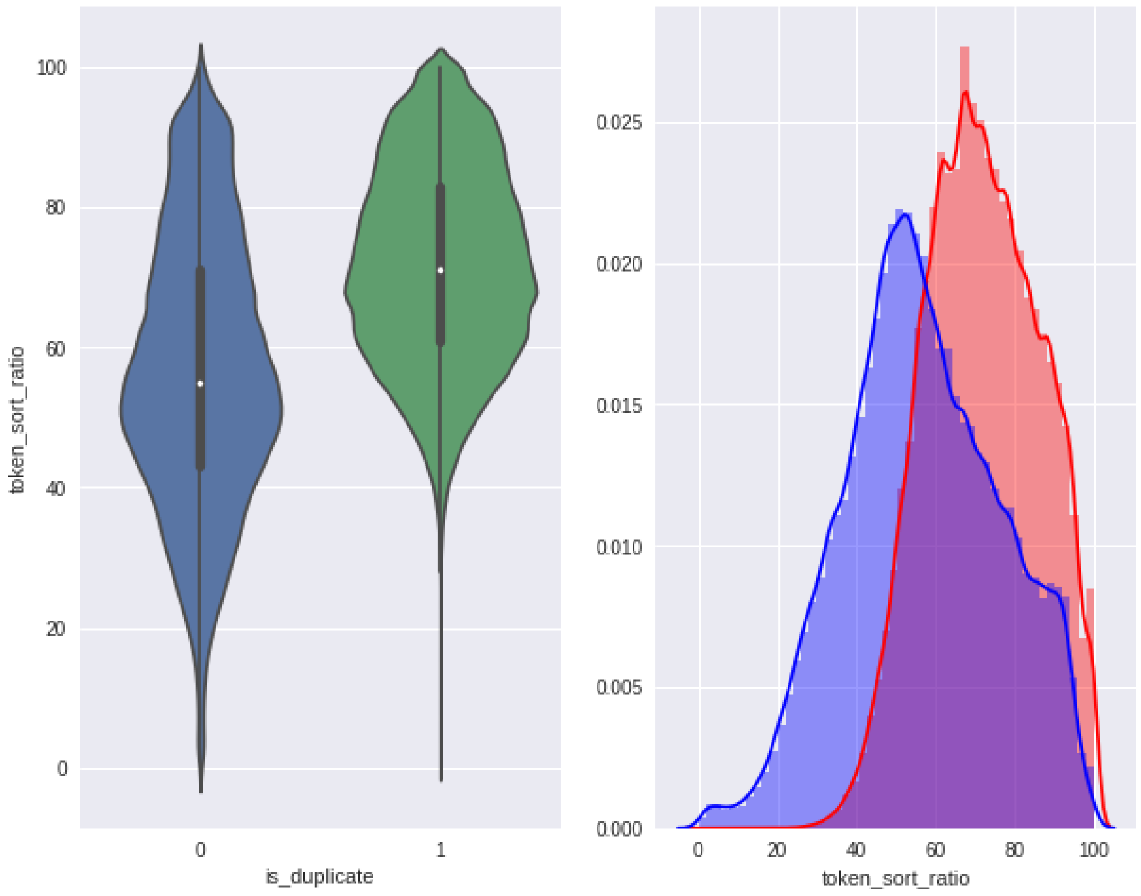
In [53]:

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



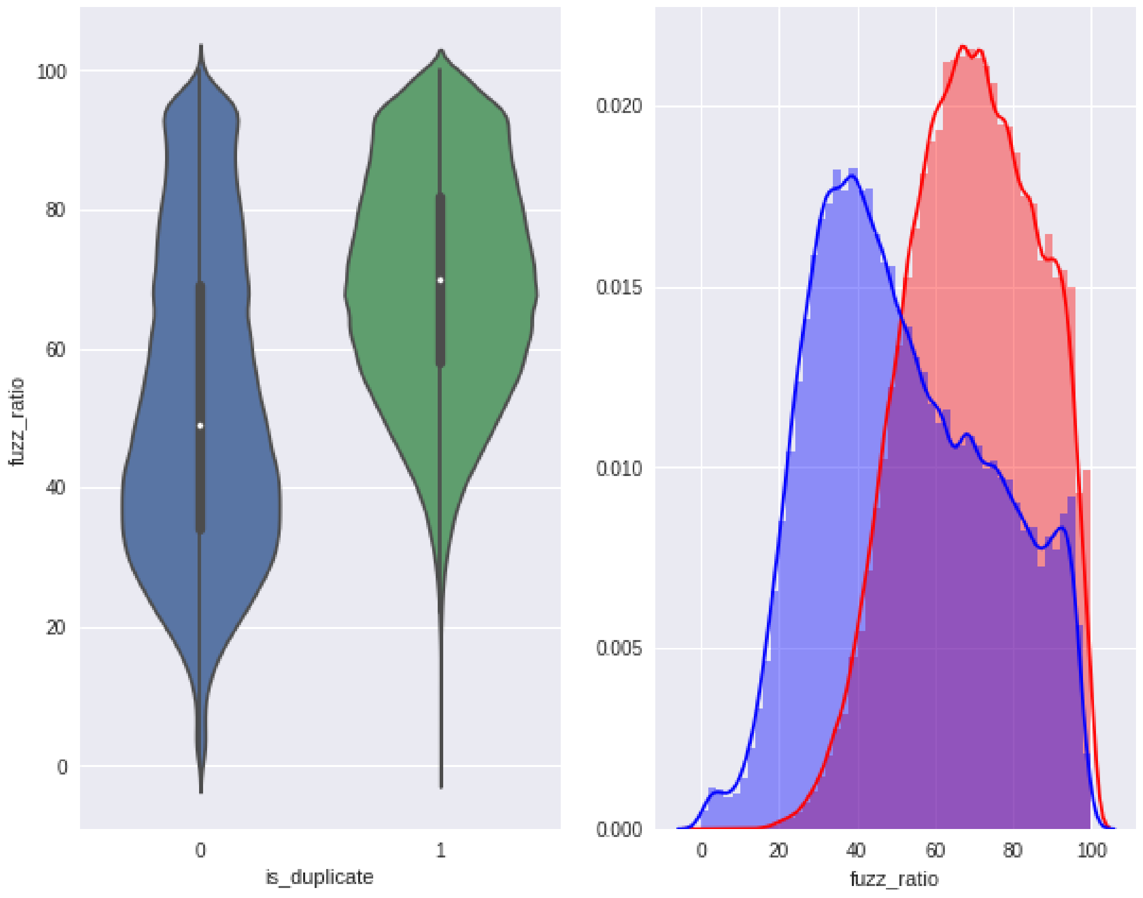
In [54]:

```
# 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" , colo
r = 'blue' )
plt.show()
```



In [55]:

```
plt.figure(figsize=(10, 8))

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

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['fuzz_ratio'][0:] , label = "1", color = 're
d')
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , color = 'b
lue' )
plt.show()
```



Combining the data:

In [0]:

```
# avoid decoding problems
df = pd.read_csv("train.csv")
```

In [0]:

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

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous no
tebook")
```

In [0]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
#df3 = df.drop(['is_duplicate'],axis=1)
```

In [59]:

```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print(df.shape[1])
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
6
```

In [60]:

```
df.head()
```

Out[60]:

	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 [61]:

```
df2.head()
```

Out[61]:

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

In [62]:

```
df1.head()
```

Out[62]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	las
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 [0]:

```
df = df.merge(df1, on='id', how = 'left')
result  = df.merge(df2, on='id',how='left')
```

In [158]:

```
result.shape
```

Out[158]:

```
(404290, 49)
```

In [0]:

```
# X is the features except is_duplicate

X = result.loc[:, result.columns != 'is_duplicate'][:100000]
y_true = result['is_duplicate'][:100000]
```

In [160]:

```
X.shape
```

Out[160]:

```
(100000, 48)
```

In [0]:

```
# loading the result and splitting to train and test
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split

# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y_true, test_size=0.33) # this i
s random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # thi
s is random splitting
```

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(X_train['question1']) + list(X_train['question2'])
tfidf = TfidfVectorizer(min_df=0.00009, max_features=2000, tokenizer = lambda text:text
.split())
tfidf.fit(questions)

X_train_qs1_tf = tfidf.transform(X_train['question1'])
X_cv_qs1_tf = tfidf.transform(X_cv['question1'])
X_test_qs1_tf = tfidf.transform(X_test['question1'])

X_train_qs2_tf = tfidf.transform(X_train['question2'])
X_cv_qs2_tf = tfidf.transform(X_cv['question2'])
X_test_qs2_tf = tfidf.transform(X_test['question2'])
```

In [0]:

```
import scipy.sparse
scipy.sparse.save_npz('X_train_qs1_tf.npz', X_train_qs1_tf)
X_train_qs1_d = scipy.sparse.load_npz('X_train_qs1_tf.npz')

scipy.sparse.save_npz('X_cv_qs1_tf.npz', X_cv_qs1_tf)
X_cv_qs1_d = scipy.sparse.load_npz('X_cv_qs1_tf.npz')

scipy.sparse.save_npz('X_test_qs1_tf.npz', X_test_qs1_tf)
X_test_qs1_d = scipy.sparse.load_npz('X_test_qs1_tf.npz')

scipy.sparse.save_npz('X_train_qs2_tf.npz', X_train_qs2_tf)
X_train_qs2_d = scipy.sparse.load_npz('X_train_qs2_tf.npz')

scipy.sparse.save_npz('X_cv_qs2_tf.npz', X_cv_qs2_tf)
X_cv_qs2_d = scipy.sparse.load_npz('X_cv_qs2_tf.npz')

scipy.sparse.save_npz('X_test_qs2_tf.npz', X_test_qs2_tf)
X_test_qs2_d = scipy.sparse.load_npz('X_test_qs2_tf.npz')
```

In [0]:

```
# remove the first row
X_train.drop(['id','qid1','qid2','question1', 'question2'], axis=1, inplace=True)
```

In [0]:

```
# dropping the unwanted columns
X_cv.drop(['id','qid1','qid2','question1', 'question2'], axis=1, inplace=True)
```

In [0]:

```
# dropping the unwanted columns
X_test.drop(['id','qid1','qid2','question1', 'question2'], axis=1, inplace=True)
```

In [0]:

```
# Combining all the features
from scipy.sparse import coo_matrix, hstack
final_train = hstack((X_train, X_train_qs1_d),format="csr",dtype='float64')

final_cv = hstack((X_cv, X_cv_qs1_d),format="csr",dtype='float64')

final_test = hstack((X_test, X_test_qs1_d),format="csr",dtype='float64')
```

Building Models:-

In [0]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predic
ted class j
    #color_codes=False
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
wo diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
wo 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, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

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

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

    plt.show()
```

1. Logistic Regression:-

Hyperparameter Tuning:

In [0]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("seaborn-darkgrid")
```

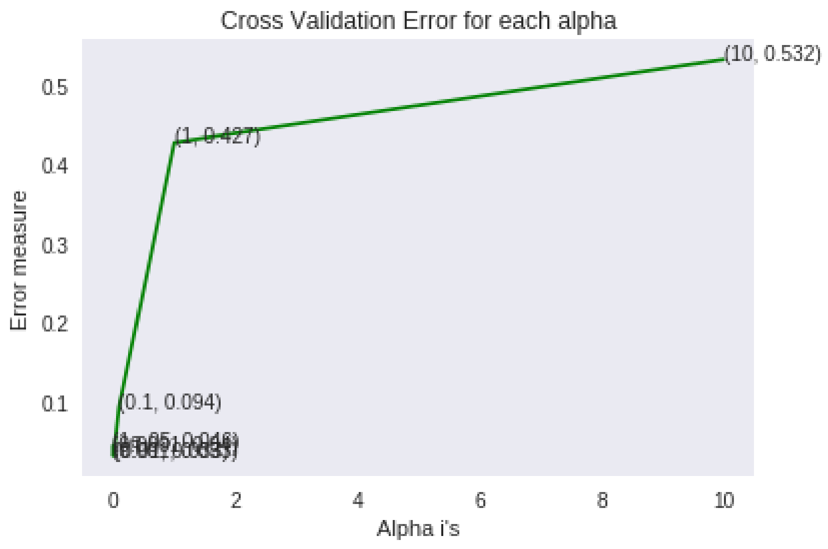
In [213]:

```
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(final_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(final_train, y_train)
    predict_y = sig_clf.predict_proba(final_cv)
    log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, lab
els=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 0.04570922619681991
For values of alpha =  0.0001 The log loss is: 0.03997297404219018
For values of alpha =  0.001 The log loss is: 0.03298135324258348
For values of alpha =  0.01 The log loss is: 0.03281184776765464
For values of alpha =  0.1 The log loss is: 0.09367947580232777
For values of alpha =  1 The log loss is: 0.4271204472459446
For values of alpha =  10 The log loss is: 0.531500485736925
```

In [214]:

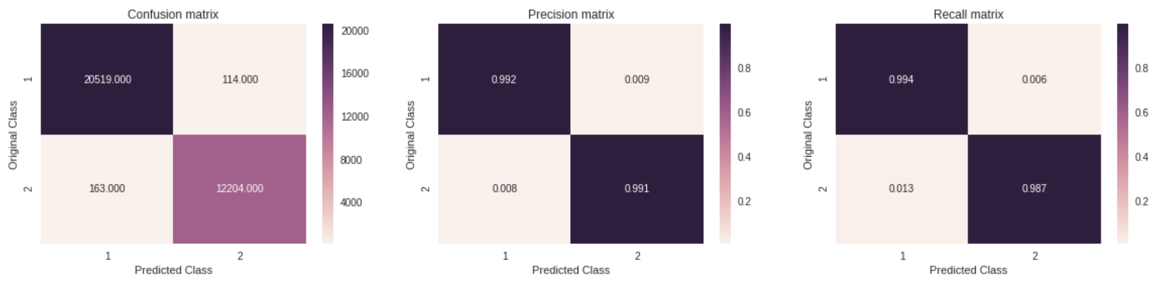
```
# plotting the error

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()
```



In [215]:

```
# testing the model

best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(final_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(final_train, y_train)

predict_y = sig_clf.predict_proba(final_cv)
print('For values of best alpha = ', alpha[best_alpha], "The CV log loss is:",log_loss(
y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(final_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(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 best alpha =  0.01 The CV log loss is: 0.03281184776765464
For values of best alpha =  0.01 The test log loss is: 0.02984221561498853
4
Total number of data points : 33000
```



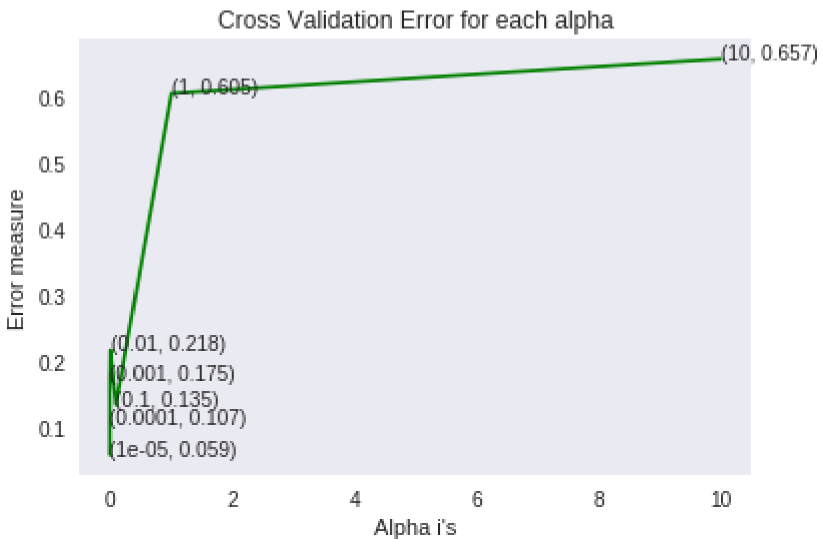
2. Linear SVM

In [217]:

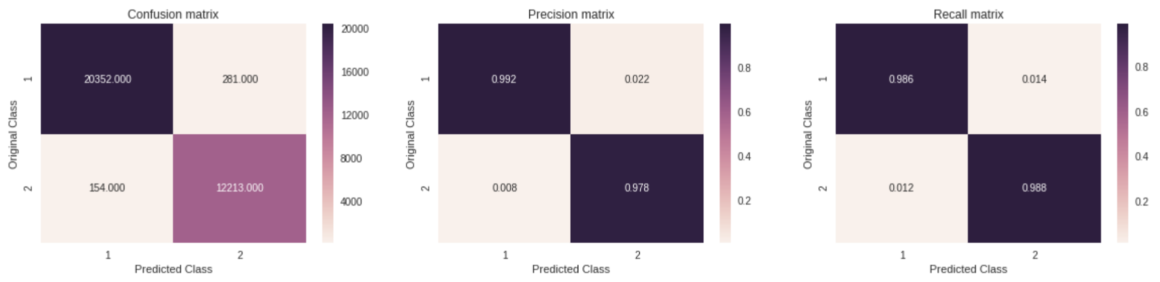
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
klearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
imal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochastic Gradie
nt Descent.
# predict(X)    Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(final_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(final_train, y_train)
    predict_y = sig_clf.predict_proba(final_cv)
    log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, lab
els=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=4
2)
clf.fit(final_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(final_train, y_train)

predict_y = sig_clf.predict_proba(final_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cv log loss is:",log_loss(
y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(final_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(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: 0.059416869249248605
For values of alpha =  0.0001 The log loss is: 0.10691951906630792
For values of alpha =  0.001 The log loss is: 0.17502956276857698
For values of alpha =  0.01 The log loss is: 0.2176505198636677
For values of alpha =  0.1 The log loss is: 0.135107708790816
For values of alpha =  1 The log loss is: 0.6050021610852733
For values of alpha =  10 The log loss is: 0.6568607475321929
```



```
For values of best alpha =  1e-05 The cv log loss is: 0.059416869249248605
For values of best alpha =  1e-05 The test log loss is: 0.0599668581781346
1
Total number of data points : 33000
```



3. XGBOOST with RandomSearch

In [221]:

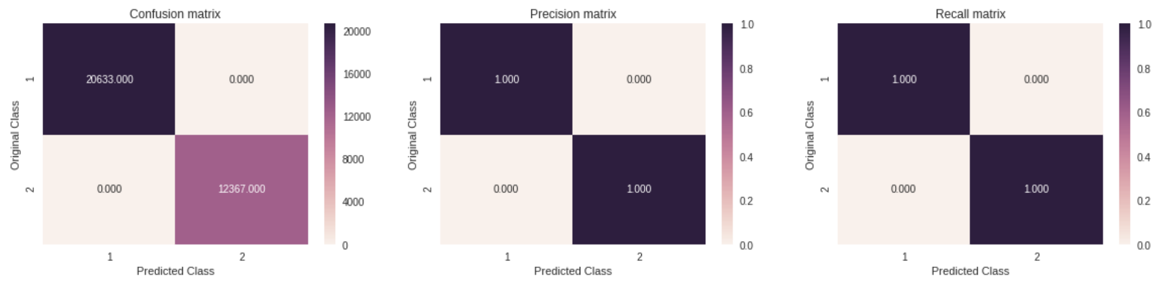
```
# code:- https://gist.github.com/wrwr/3f6b66bf4ee01bf48be965f60d14454d

import time

import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV

clf = xgb.XGBClassifier()

param_grid = {
        'silent': [False],
        'max_depth': [6, 10, 15, 20],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
        'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'colsample_bylevel': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
        'gamma': [0, 0.25, 0.5, 1.0],
        'reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
        'n_estimators': [100]}

rs_clf = RandomizedSearchCV(clf, param_grid, n_iter=20,n_jobs=-1, verbose=2, scoring='n
eg_log_loss', refit=False, random_state=42)

print("Randomized search..")
search_time_start = time.time()
rs_clf.fit(final_train, y_train)
print("Randomized search time:", time.time() - search_time_start)

best_score = rs_clf.best_score_
best_params = rs_clf.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
  print('%s: %r' % (param_name, best_params[param_name]))
```

```
Randomized search..
Fitting 3 folds for each of 20 candidates, totalling 60 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  2.9min

Randomized search time: 274.2403509616852
Best score: -4.6320422001683005e-05
Best params:
colsample_bylevel: 1.0
colsample_bytree: 1.0
gamma: 0
learning_rate: 3
max_depth: 10
min_child_weight: 0.5
n_estimators: 100
reg_lambda: 50.0
silent: False
subsample: 0.7

[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:  4.6min finished
```

In [229]:

```
import xgboost as XG
XGBClass = XG.XGBClassifier(learning_rate=3, n_estimators=100,nthread=-1)

XGBClass.fit(final_train, y_train)

predict_y = XGBClass.predict_proba(final_cv)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_cv, predict_y, eps=1e-15))
predict_y = XGBClass.predict_proba(final_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(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)
```

```
For values of best alpha =  1e-05 The train log loss is: 1.385453908525017
4e-05
For values of best alpha =  1e-05 The test log loss is: 1.3894348211496519
e-05
Total number of data points : 33000
```



In [232]:

```
from prettytable import PrettyTable
ptable = PrettyTable()

ptable.title = " Comparison of Performances "
ptable.field_names = ['Model','Hyperparameter', 'CV log loss ','Test Log Loss']
ptable.add_row(["Logistic Regression","0.01","0.032", "0.029"])
ptable.add_row(["Linear SVM", "1e-05", "0.059", "0.059"])
ptable.add_row(["XGBOOST","1e-05","1.3e-05", "1.38e-05"])

print(ptable)
```

```
+---------------------+----------------+--------------+---------------+
|        Model        | Hyperparameter | CV log loss  | Test Log Loss |
+---------------------+----------------+--------------+---------------+
| Logistic Regression |      0.01      |    0.032     |     0.029     |
|      Linear SVM     |     1e-05      |    0.059     |     0.059     |
|       XGBOOST       |     1e-05      |   1.3e-05    |    1.38e-05   |
+---------------------+----------------+--------------+---------------+
```

Few steps followed in the assignment :

1. Initially we have performed EDA on the dataset and found many good insights like total questions, no of duplicate questions, non-duplicate etc.
2. In addition to that i have added two more EDA of no. of words in both questions combined cell[10] and total no of characters in questions cell[17].
3. After preprocessing part, i loaded the dataset with nlp features and basic features before preprocessing and combined them.
4. I have done train test split on the features by considering 100k points named them, X_train, X_test, X_cv.
5. Extracted the X_train['question1'] and X_train['question2'], similarly for X_cv and X_train.
6. Applied Tfidffeaturization on X_train['question1'] and X_train['question2'].
7. Initially i combined X_train['question1'] and X_train['question2'] and fitted the classifier and then transformed the learned classifier individually on the splitted datsets.
8. After that i appended my old dataset with the featurized sparse matrix and appended them using hstack in sparse format.
9. After that i trained my 3 models individually and got the above result.
10. Among them i can observe that XGBOOST performed the best.