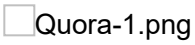


Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

Problem Statement

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

1.2 Sources/Useful Links

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

Useful Links

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments (https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments)

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

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

```
"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) Diamond?","What w
ould happen if the Indian government stole the Kohinoor (Koh-i-Noor) diam
ond 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 s
ee 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 [1]:
```
!pip install distance
!pip install fuzzywuzzy
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437aefa0
d0a92ccfb38895d9416bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz (https://files.
pythonhosted.org/packages/5c/1a/883e47df323437aefa0d0a92ccfb38895d9416bd0b56262
c2e46a47767b8/Distance-0.1.3.tar.gz) (180kB)
    100% |████████████████████████████████| 184kB 31.1MB/s a 0:00:01
Building wheels for collected packages: distance
  Running setup.py bdist_wheel for distance ... done
  Stored in directory: /home/j_choudhary1001/.cache/pip/wheels/d5/aa/e1/dbba9e7
b6d397d645d0f12db1c66dbae9c5442b39b001db18e
Successfully built distance
distributed 1.21.8 requires msgpack, which is not installed.
Installing collected packages: distance
Successfully installed distance-0.1.3
You are using pip version 10.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7eaa1b
eab2b9323073815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-none-any.wh
l (https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7eaa1beab2b932307
3815da4551076554ecc890a3595ec9/fuzzywuzzy-0.17.0-py2.py3-none-any.whl)
distributed 1.21.8 requires msgpack, which is not installed.
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
You are using pip version 10.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

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

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings("ignore")
import 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 two strings
# 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-wordcloud-in-python
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image

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
import spacy

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
```

3.1 Reading data and basic stats

In [4]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:
```
df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/train.csv")

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

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

In [0]:
```
df.head()
```

Out[8]:

	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 [0]:
```
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:

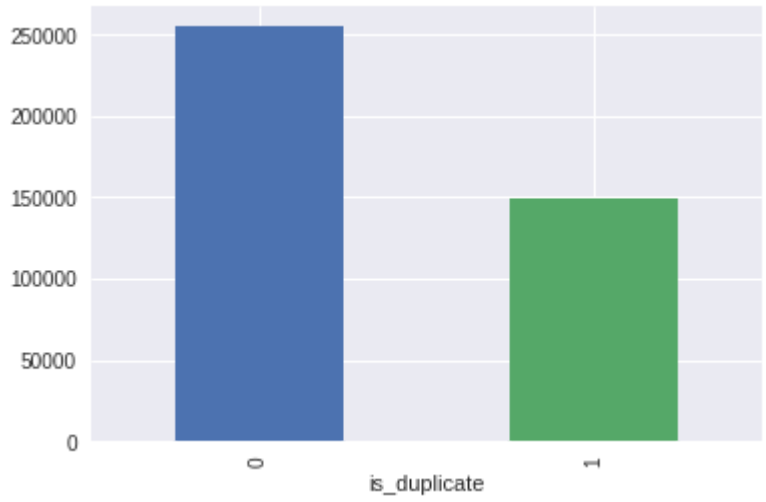
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

In [0]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb628d3c3c8>
```



In [0]:
```
print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

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

In [0]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(
```

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

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

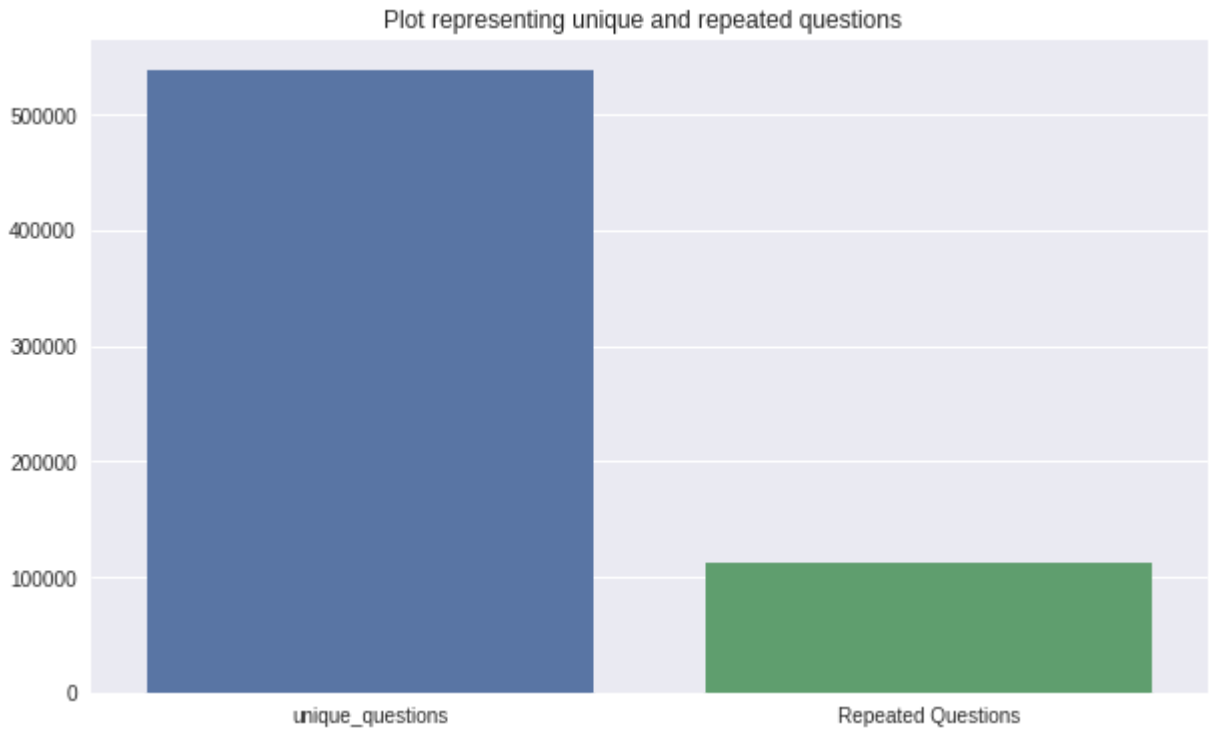
3.2.2 Number of unique questions

In [0]:
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {} ({}%)\n'.fo

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

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 (20.779539459
37505%)

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

In [0]:
```
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:1428: FutureWarni
ng:

remove_na is deprecated and is a private function. Do not use.
```



3.2.3 Checking for Duplicates

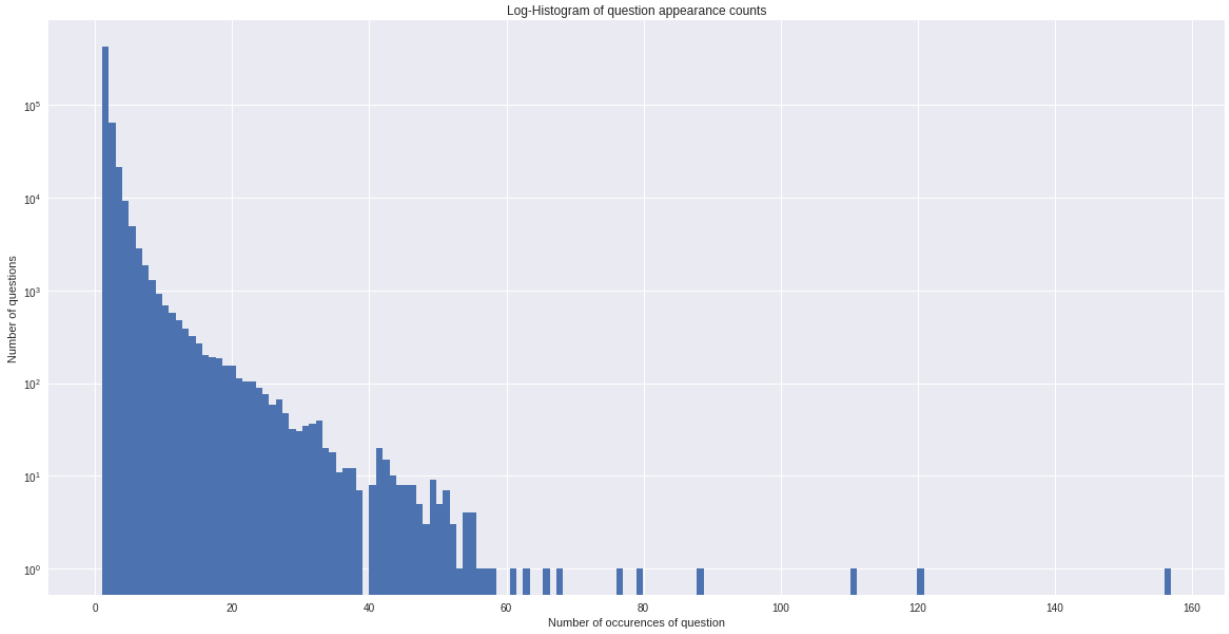
In [0]:
```
#checking whether there are any repeated pair of questions

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

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 [0]:
```
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(q
```

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



3.2.5 Checking for NULL values

In [0]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

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

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

- There are two rows with null values in question2

In [0]:
```
# 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 [0]:
```
if os.path.isfile('/content/gdrive/My Drive/Colab Notebooks/Quora/df_fe_without_p
    df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/df_fe_withou
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("/content/gdrive/My Drive/Colab Notebooks/Quora/df_fe_without_prepr

df.head()
```

Out[19]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_v
0	0	1	2	What is the step by step guide to invest in 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	

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_v
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.

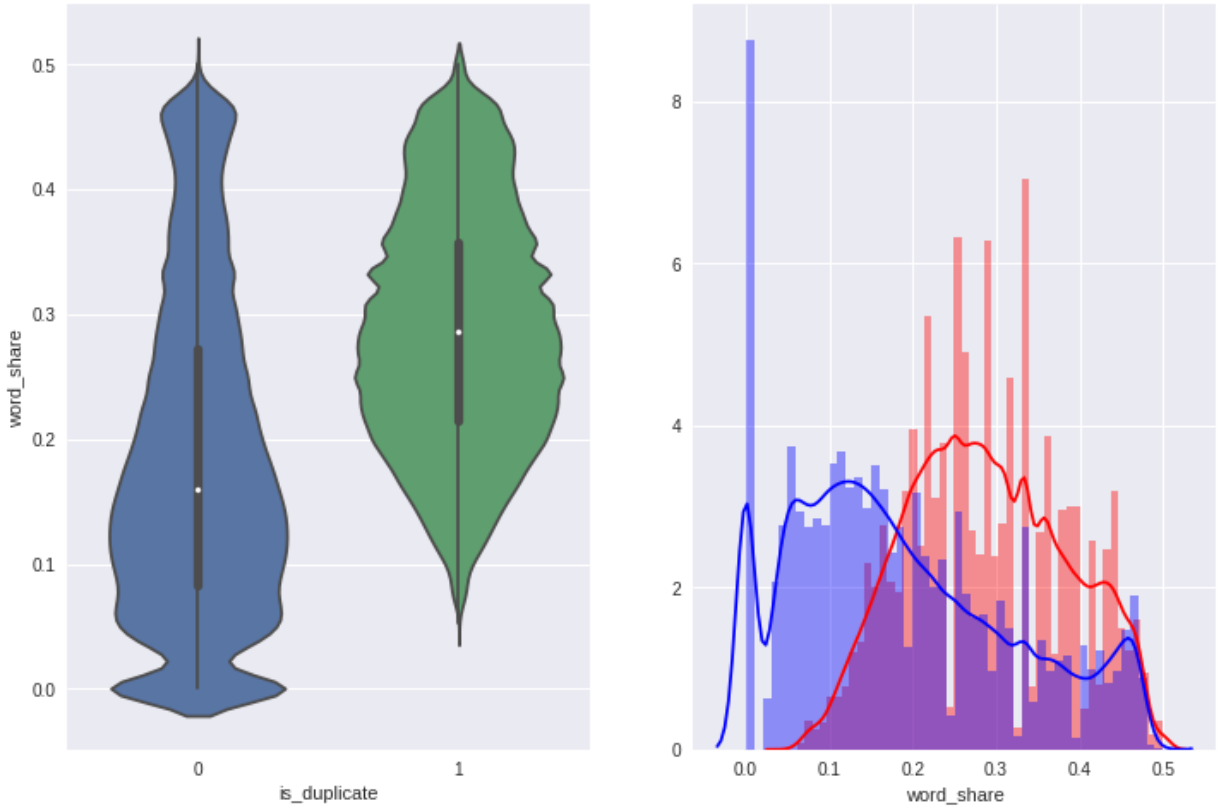
In [0]:
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words
```

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g:

remove_na is deprecated and is a private function. Do not use.

/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g:

remove_na is deprecated and is a private function. Do not use.

/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: Matplotli
bDeprecationWarning:


The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1.
Use 'density' instead.
```



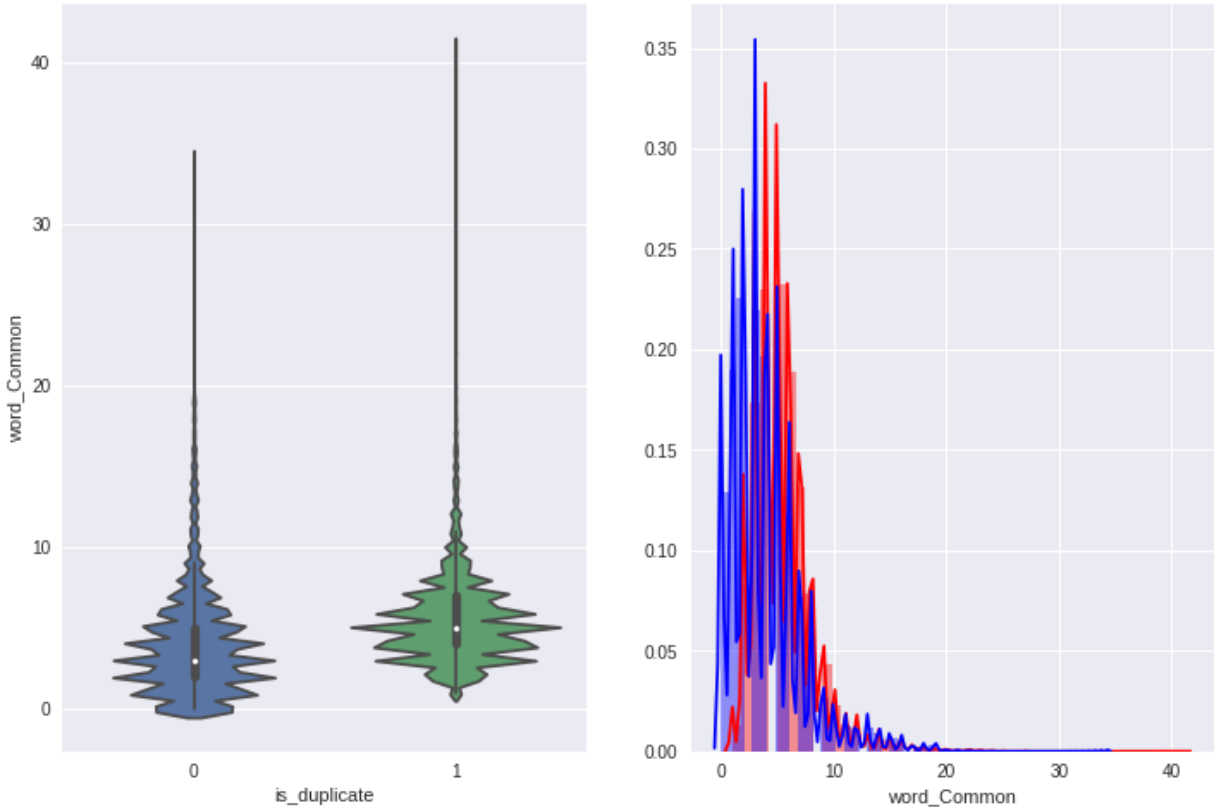
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g:

remove_na is deprecated and is a private function. Do not use.

/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g:

remove_na is deprecated and is a private function. Do not use.

/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: Matplotli
bDeprecationWarning:


The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1.
Use 'density' instead.
```



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

In [0]:

```
```

In [0]:

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-
if os.path.isfile('/content/gdrive/My Drive/Colab Notebooks/Quora/df_fe_without_p
    df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/df_fe_withou
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previo
```

In [0]:

```
df.head(2)
```

Out[12]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_wo
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	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	

In [8]:

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

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/j_choudhary1001/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[8]:

```
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 n
                           .replace("n't", " not").replace("what's", "what is").r
                           .replace("'ve", " have").replace("i'm", "i am").replac
                           .replace("he's", "he is").replace("she's", "she is").r
                           .replace("%", " percent ").replace("₹", " rupee ").rep
                           .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
```

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)) + 
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + 
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + 
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + 
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens))
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens))

    # 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["que

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

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["ques
    # The token sort approach involves tokenizing the string in question, sorting
    # then joining them back into a string We then compare the transformed string
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["que
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"],
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["questi
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["
    return df
```

In [7]:
```
if os.path.isfile('/content/gdrive/My Drive/Colab Notebooks/Quora/nlp_features_tr
    df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/nlp_features_
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/train.csv")
    df = extract_features(df)
    df.to_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/nlp_features_train.
df.head(2)
```

Out[7]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	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	0.599988	...

2 rows × 21 columns

In [8]:
```
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 {
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).fla

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')
```

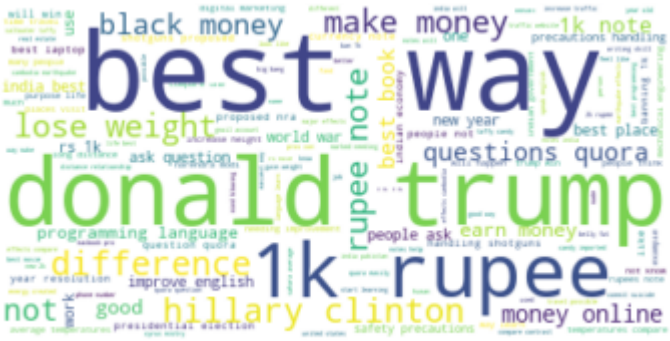
```
Number of data points in class 1 (duplicate pairs) : 298526
Number of data points in class 0 (non duplicate pairs) : 510054
```

In [9]:
```
# 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 : 33193130
```

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

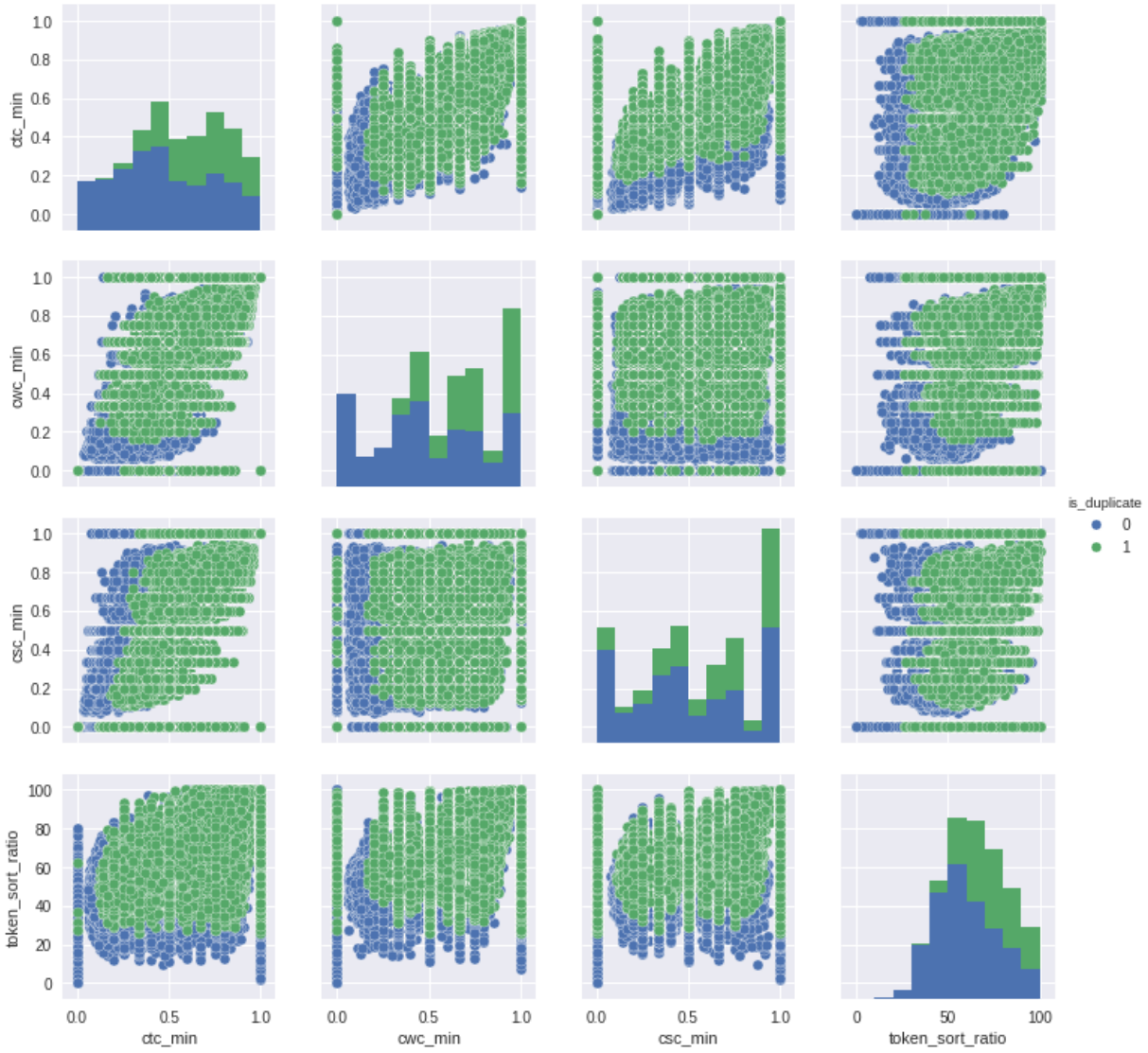
```
Word Cloud for Duplicate Question pairs
```



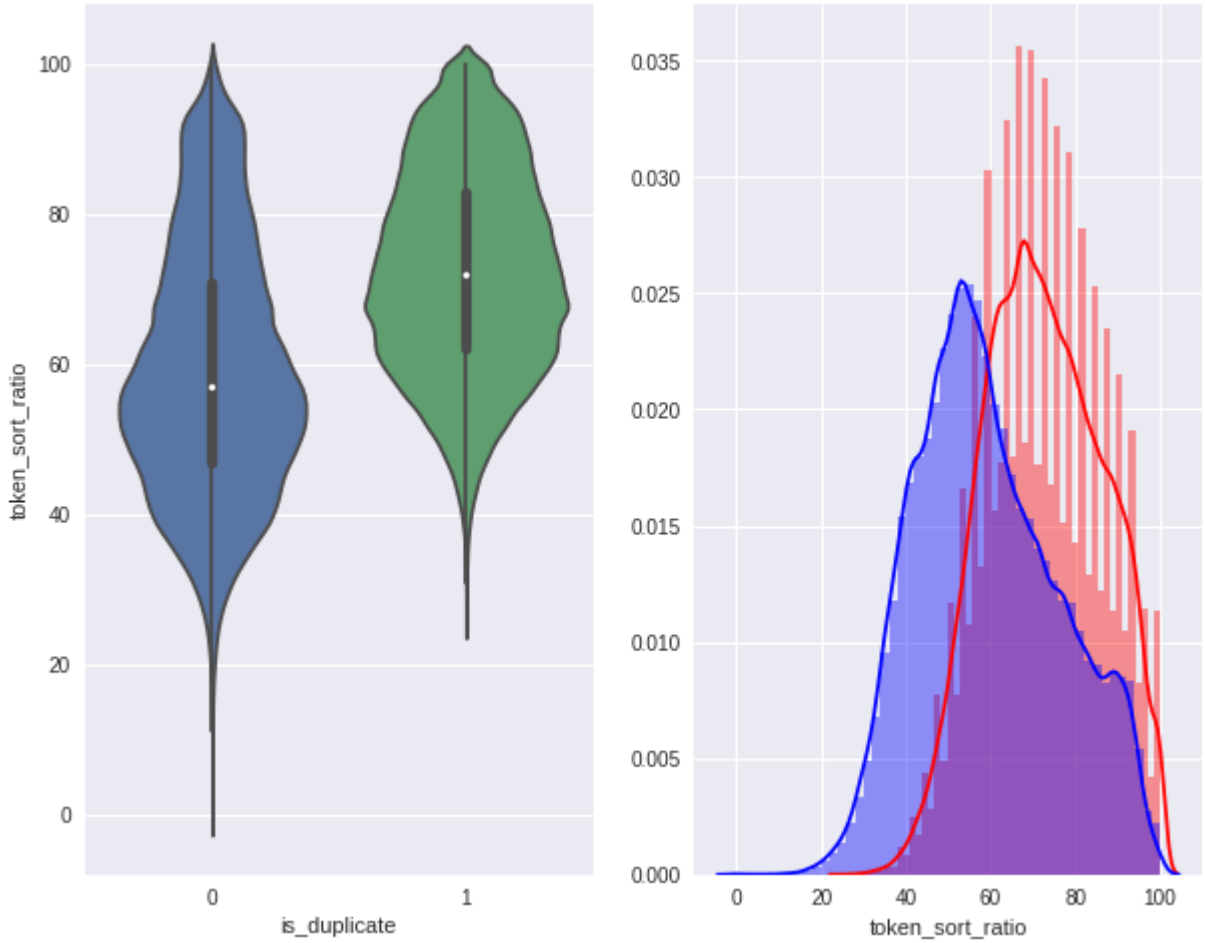
In [11]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwor
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

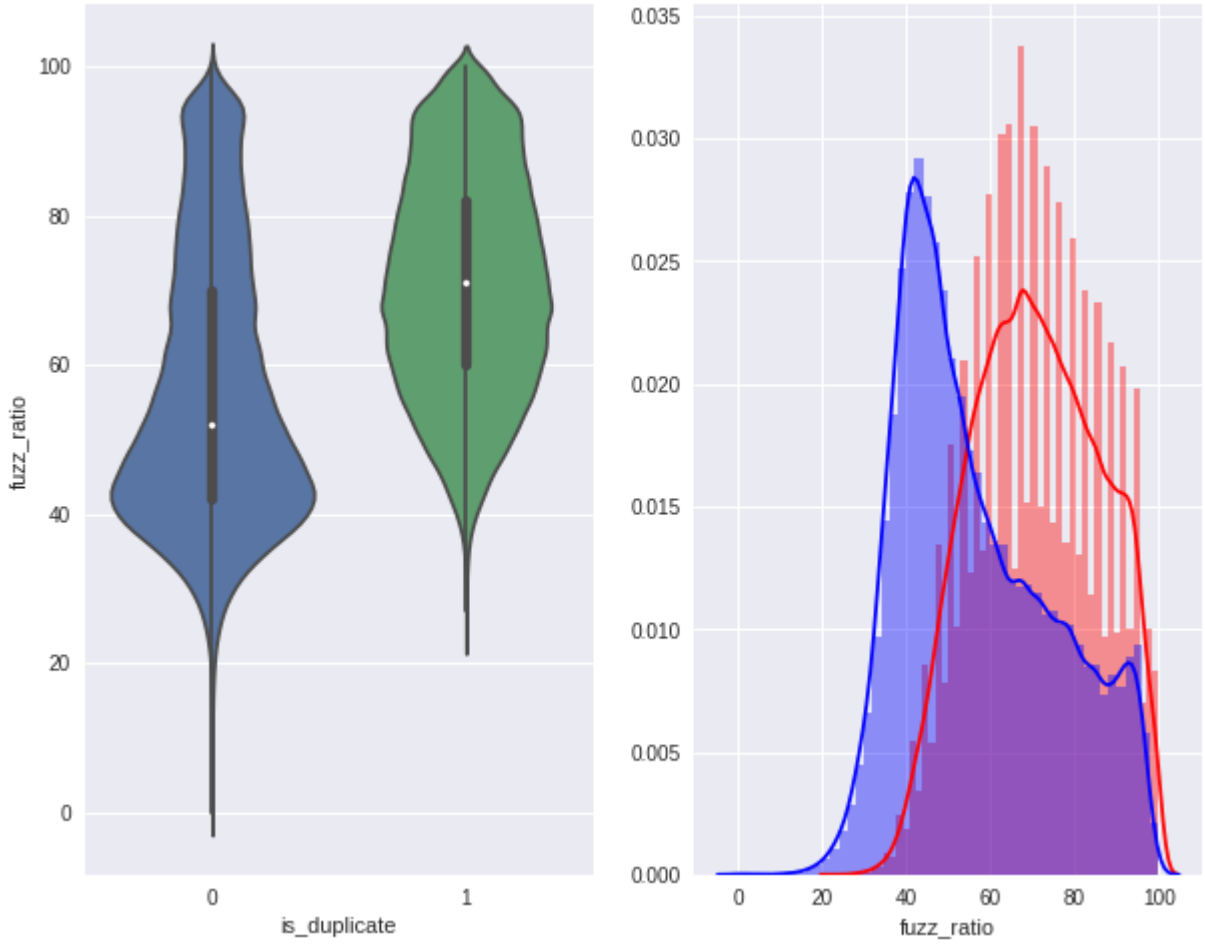
In [12]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplica
plt.show()
```



In [13]:
```
# 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",
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0"
plt.show()
```



In [14]:
```
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
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , colo
plt.show()
```



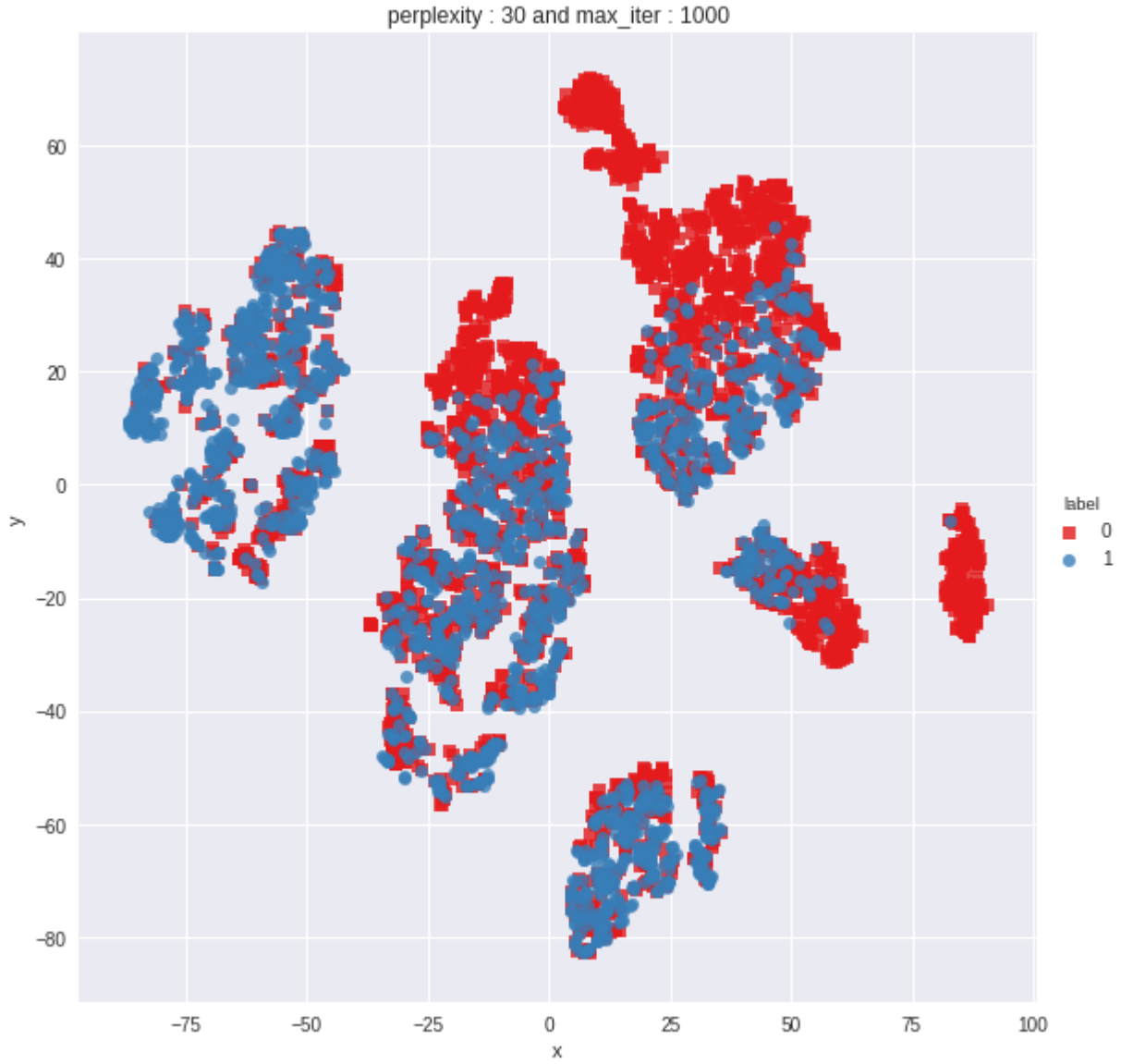
In [0]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleanin

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min',
y = dfp_subsampled['is_duplicate'].values
```

In [16]:
```
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 0.017s...
[t-SNE] Computed neighbors for 5000 samples in 0.371s...
[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 0.308s
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iterati
ons in 2.701s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iterat
ions in 2.005s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iterat
ions in 1.980s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iterat
ions in 2.048s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iterat
ions in 2.095s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.273346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 iterati
ons in 2.095s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iterati
ons in 2.047s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iterati
ons in 2.057s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iterati
ons in 2.085s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iterati
ons in 2.080s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iterati
ons in 2.074s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iterati
ons in 2.092s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iterati
ons in 2.089s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iterati
ons in 2.095s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iterati
ons in 2.088s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iterati
ons in 2.090s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iterati
ons in 2.079s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iterati
ons in 2.075s)
```

```
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iterati
ons in 2.095s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iterat
ions in 2.111s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

In [17]:
```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

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



In [18]:
```
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 0.015s...
[t-SNE] Computed neighbors for 5000 samples in 0.374s...
[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 0.302s
[t-SNE] Iteration 50: error = 80.3552017, gradient norm = 0.0329941 (50 iterati
ons in 13.168s)
[t-SNE] Iteration 100: error = 69.1127167, gradient norm = 0.0036756 (50 iterat
ions in 6.816s)
[t-SNE] Iteration 150: error = 67.6178818, gradient norm = 0.0017629 (50 iterat
ions in 6.043s)
[t-SNE] Iteration 200: error = 67.0571747, gradient norm = 0.0011826 (50 iterat
ions in 5.997s)
[t-SNE] Iteration 250: error = 66.7298050, gradient norm = 0.0008528 (50 iterat
ions in 5.863s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729805
[t-SNE] Iteration 300: error = 1.4963876, gradient norm = 0.0006857 (50 iterati
ons in 8.084s)
[t-SNE] Iteration 350: error = 1.1549060, gradient norm = 0.0001911 (50 iterati
ons in 10.753s)
[t-SNE] Iteration 400: error = 1.0083323, gradient norm = 0.0000968 (50 iterati
ons in 10.749s)
[t-SNE] Iteration 450: error = 0.9356370, gradient norm = 0.0000660 (50 iterati
ons in 10.781s)
[t-SNE] Iteration 500: error = 0.8982100, gradient norm = 0.0000521 (50 iterati
ons in 10.544s)
[t-SNE] Iteration 550: error = 0.8778670, gradient norm = 0.0000595 (50 iterati
ons in 10.442s)
[t-SNE] Iteration 600: error = 0.8642665, gradient norm = 0.0000579 (50 iterati
ons in 10.509s)
[t-SNE] Iteration 650: error = 0.8558875, gradient norm = 0.0000362 (50 iterati
ons in 10.461s)
[t-SNE] Iteration 700: error = 0.8492573, gradient norm = 0.0000305 (50 iterati
ons in 10.712s)
[t-SNE] Iteration 750: error = 0.8432317, gradient norm = 0.0000276 (50 iterati
ons in 10.743s)
[t-SNE] Iteration 800: error = 0.8378869, gradient norm = 0.0000279 (50 iterati
ons in 10.624s)
[t-SNE] Iteration 850: error = 0.8331724, gradient norm = 0.0000261 (50 iterati
ons in 10.462s)
[t-SNE] Iteration 900: error = 0.8291837, gradient norm = 0.0000259 (50 iterati
```

```
ons in 10.260s)
[t-SNE] Iteration 950: error = 0.8255505, gradient norm = 0.0000333 (50 iterati
ons in 10.251s)
[t-SNE] Iteration 1000: error = 0.8224180, gradient norm = 0.0000235 (50 iterat
ions in 10.191s)
[t-SNE] KL divergence after 1000 iterations: 0.822418
```

In [19]:
```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

data=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered features')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

In [35]:
```
# avoid decoding problems
# df = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/Quora/train.csv")
df = pd.read_csv("train.csv")
# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
# ----------------- python 3 ---------------------
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [36]:
```
df.head()
```

Out[36]:

	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 [37]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts

# question1 = pd.concat([df['question1'], df['question2']], axis=1)
questions1 = df['question1']
questions2 = df['question2']
y = df['is_duplicate']
```

In [61]:
```
y_train = y[:80000]
# y_val = y[60000:80000]
y_test = y[80000:100000]
# y_train = y[:160000]
# y_test = y[160000:200000]
```

In [62]:
```
len(y_train)
```

Out[62]:
```
80000
```

In [41]:
```
# X_train, X_test, y_train, y_test = train_test_split(questions, y, test_size=0.3
q1_train = questions1[:80000]
q1_test = questions1[80000:100000]
q2_train = questions2[:80000]
# q2_val = questions2[60000:80000]
q2_test = questions2[80000:100000]

questions = list(q1_train) + list(q2_train)
# ques_train = q1_train.values.tolist() + q2_train.values.tolist()
# ques_test = q1_test.values.tolist() + q2_test.values.tolist()
```

In [42]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts

tfidf = TfidfVectorizer(lowercase=False,)
tfidf.fit(questions)
q1_train = tfidf.transform(q1_train.values.tolist())
q2_train = tfidf.transform(q2_train.values.tolist())
# q1_val = tfidf.transform(q1_val.values.tolist())
# q2_val = tfidf.transform(q2_val.values.tolist())
q1_test = tfidf.transform(q1_test.values.tolist())
q2_test = tfidf.transform(q2_test.values.tolist())
# q_tfidf_test = tfidf.transform(ques_test)


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

In [43]:
```
print(q1_train.shape)
print(q2_train.shape)
# print(q1_val.shape)
# print(q2_val.shape)
print(q1_test.shape)
print(q2_test.shape)
# print(q_tfidf_train.shape)
# print(q_tfidf_test.shape)
```

```
(80000, 50964)
(80000, 50964)
(20000, 50964)
(20000, 50964)
```

In [44]:
```
train_questions = hstack((q1_train, q2_train))
# val_questions = hstack((q1_val, q2_val))
test_questions = hstack((q1_test, q2_test))
```

In [45]:
```
print(train_questions.shape)
# print(val_questions.shape)
print(test_questions.shape)
# print(type(q_tfidf_train))
```

```
(80000, 101928)
(20000, 101928)
```

In [13]:
```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
# if os.path.isfile('nlp_features_train.csv')
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-
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previ
```

In [14]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [15]:
```
df1.head()
```

Out[15]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first
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 [16]:
```
df2.head()
```

Out[16]:

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

In [46]:
```
import scipy
df1_sparse_train = scipy.sparse.csr_matrix(df1[:80000].to_sparse())
# df1_sparse_val = scipy.sparse.csr_matrix(df1[60000:80000].to_sparse())
df1_sparse_test = scipy.sparse.csr_matrix(df1[80000:100000].to_sparse())
df2_sparse_train = scipy.sparse.csr_matrix(df2[:80000].to_sparse())
# df2_sparse_val = scipy.sparse.csr_matrix(df2[60000:80000].to_sparse())
df2_sparse_test = scipy.sparse.csr_matrix(df2[80000:100000].to_sparse())
# df1_sparse_train = scipy.sparse.csr_matrix(df1[:160000].to_sparse())
# df1_sparse_test = scipy.sparse.csr_matrix(df1[160000:200000].to_sparse())
# df2_sparse_train = scipy.sparse.csr_matrix(df2[:160000].to_sparse())
# df2_sparse_test = scipy.sparse.csr_matrix(df2[160000:200000].to_sparse())
```

In [47]:
```
print(type(df1_sparse_train))
print(type(df2_sparse_train))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

In [48]:
```
df1_sparse_train
```

Out[48]:
```
<80000x17 sparse matrix of type '<class 'numpy.float64'>'
        with 1158909 stored elements in Compressed Sparse Row format>
```

In [49]:
```
print(df1_sparse_train.shape)
print(df1_sparse_test.shape)
print(df2_sparse_train.shape)
print(df2_sparse_test.shape)
```

```
(80000, 17)
(20000, 17)
(80000, 12)
(20000, 12)
```

In [50]:
```
df_train = hstack((df1_sparse_train, df2_sparse_train))
# df_val = hstack((df1_sparse_val, df2_sparse_val))
df_test = hstack((df1_sparse_test, df2_sparse_test))
```

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

Out[51]:
```
(80000, 29)
```

In [52]:
```
final_train = hstack((train_questions, df_train))
# final_val = hstack((val_questions, df_val))
final_test = hstack((test_questions, df_test))
# final_train = hstack((q_tfidf_train, df_train))
# final_test = hstack((q_tfidf_test, df_test))
```

In [53]:
```
print(final_train.shape)
print(final_test.shape)
```

```
(80000, 101957)
(20000, 101957)
```

In [54]:
```
from scipy import sparse
```

In [108]:
```
from scipy.sparse import hstack

# storing the final features to csv f_ile
if not os.path.isfile('final_features_tfidf.csv'):
#     df1_sparse['id']=df1['id']
#     df2_sparse['id']=df1['id']
#     df1  = df1.merge(df2, on='id',how='left')
#     df2  = df1_sparse.merge(df2_sparse, on='id',how='left')
#     result  = df1.merge(df2, on='id',how='left')
#     result = hstack((tfidf_train, features_df))
    final_train.to_csv('final_features_tfidf.csv')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-108-70acad1706c2> in <module>()
      9 #     result  = df1.merge(df2, on='id',how='left')
     10 #     result = hstack((tfidf_train, features_df))
---> 11     final_train.to_csv('final_features_tfidf.csv')

~/anaconda3/lib/python3.6/site-packages/scipy/sparse/base.py in __getattr__(sel
f, attr)
    684             return self.getnnz()
    685         else:
--> 686             raise AttributeError(attr + " not found")
    687
    688     def transpose(self, axes=None, copy=False):

AttributeError: to_csv not found
```

In [26]:
```
# dataframe of nlp features
df1.head()
```

Out[26]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first
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]:
```
# data before preprocessing
df2.head()
```

Out[55]:

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

In [ ]:
```
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 final dataframe  :", df1.shape[1]+df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape[1])
print("Number of features in question2 w2v  dataframe :", df3_q2.shape[1])
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df3_q
```

In [ ]:
```
# storing the final features to csv file
if not os.path.isfile('final_features_tfidf.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 [ ]:
```
# 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 [27]:
```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
import sqlite3
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 [28]:
```
# read_db = '/content/gdrive/My Drive/Colab Notebooks/Quora/train.db'
read_db = 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()

# import pandas as pd

# conn = sqlite3.connect(read_db)

# cursor = conn.cursor()

# cursor.execute("select name from sqlite_master where type='table'")

# df=pd.DataFrame(cursor.fetchall())
```

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

In [29]:
```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)

        # for selecting random points
        data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 1000
        conn_r.commit()
        conn_r.close()
```

In [30]:
```
# remove the first row
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace=True)
```

In [31]:
```
data.head()
```

Out[31]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_mi
1	0.999950002499875	0.66664444518516	0.33332222259258	0.249993750156246	0.49999166680555
2	0.799984000319994	0.799984000319994	0.624992187597655	0.49999500005	0.6923023669048
3	0.499975001249937	0.33332222259258	0.499975001249937	0.33332222259258	0.39999200015999
4	0.66664444518516	0.19999800002	0.999966667777741	0.272724793410969	0.83331944467592
5	0.66664444518516	0.499987500312492	0.749981250468738	0.428565306209911	0.62499218759765

5 rows × 794 columns

In [32]:
```
cols = list(data.columns)
```

In [33]:
```
len(cols)
```

Out[33]: 794

In [ ]:
```
# 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)
```

In [ ]:
```
# 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)
```

In [0]:
```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-in
y_true = list(map(int, y_true.values))
```

In [116]:
```
len(y_train)
```

Out[116]:
```
80000
```

In [56]:
```
print("Number of data points in train data :",final_train.shape)
print("Number of data points in test data :",final_test.shape)
```

```
Number of data points in train data : (80000, 101957)
Number of data points in test data : (20000, 101957)
```

In [25]:
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/tes
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.627475 Class 1:  0.372525
---------- Distribution of output variable in train data ----------
Class 0:  0.3726 Class 1:  0.3726
```

In [57]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are p

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

    # 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
    # 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 c
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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.color_palette()
    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()
```

In [64]:
```
C = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
train_score_list = []
test_score_list = []
for i in C:
    clf=LogisticRegression(C=i, penalty='l2')
    clf.fit(final_train,y_train)
    train_score = clf.score(final_train, y_train)
    train_score_list.append(train_score)
    test_scores = clf.score(final_test, y_test)
    test_score_list.append(test_scores)
    test_score = clf.predict_proba(final_test)
    log_error_array.append(log_loss(y_test, test_score, labels=clf.classes_, eps=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, test_s
```

```
For values of alpha =  1e-05 The log loss is: 0.49446704501309957
For values of alpha =  0.0001 The log loss is: 0.43298030643785207
For values of alpha =  0.001 The log loss is: 0.11459608038471197
For values of alpha =  0.01 The log loss is: 0.16361760376160606
For values of alpha =  0.1 The log loss is: 0.16721079020404842
For values of alpha =  1 The log loss is: 0.06975395712077066
For values of alpha =  10 The log loss is: 0.09341066839977531
```

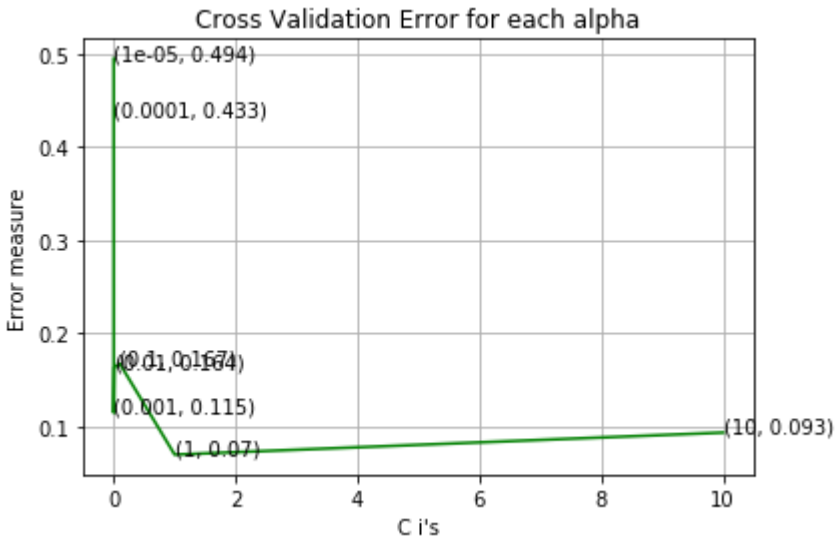
In [65]:
```
tfidf_val_score = dict(zip(C, test_score_list))
best_c = max(tfidf_val_score, key=tfidf_val_score.get)
best_c
```

Out[65]: 1

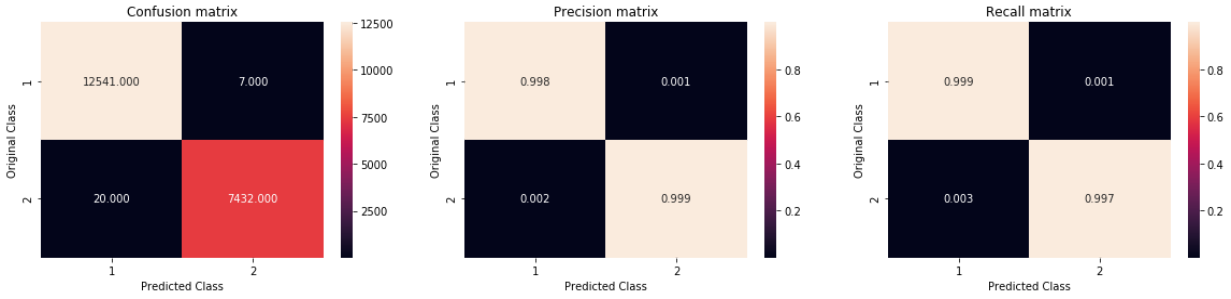
In [66]:
```
clf=LogisticRegression(C=best_c, penalty='l2')
clf.fit(final_train,y_train)
```

Out[66]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [69]:
```
fig, ax = plt.subplots()
ax.plot(C, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((C[i],np.round(txt,3)), (C[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("C i's")
plt.ylabel("Error measure")
plt.show()
```



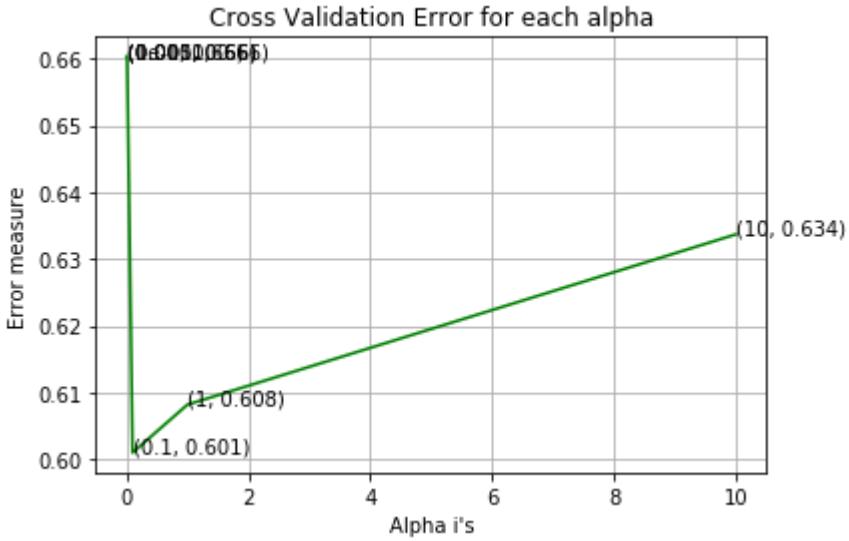
In [70]:
```
test_probabilities = clf.predict_proba(final_test)
predicted_y =np.argmax(test_probabilities,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```



In [71]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42, cla
    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_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predic
```

```
For values of alpha =  1e-05 The log loss is: 0.6603249672711292
For values of alpha =  0.0001 The log loss is: 0.6603249672711292
For values of alpha =  0.001 The log loss is: 0.6603249672711292
For values of alpha =  0.01 The log loss is: 0.6603249672711292
For values of alpha =  0.1 The log loss is: 0.6010065723116299
For values of alpha =  1 The log loss is: 0.6081928624803153
For values of alpha =  10 The log loss is: 0.6337041686964044
```

In [72]:
```
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()
```

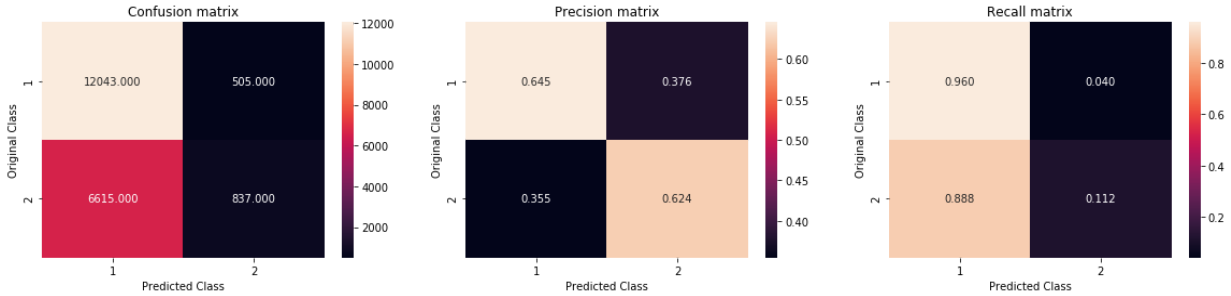


In [73]:
```
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_s
clf.fit(final_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(final_train, y_train)
```

Out[73]:
```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.1, average=False, c
lass_weight='balanced', epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l1', power_t=0.5, random_state=42, shuffle=True,
       tol=None, verbose=0, warm_start=False),
            cv=3, method='sigmoid')
```

In [74]:
```
predict_y = sig_clf.predict_proba(final_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",
predict_y = sig_clf.predict_proba(final_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
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 =  0.1 The train log loss is: 0.6006785544260158
For values of best alpha =  0.1 The test log loss is: 0.6010065723116299
Total number of data points : 20000
```



In [75]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV

# tuned_parameters ={'objective': 'binary:logistic',
#                      'eval_metric': 'logloss',
#                    'eta': 0.02,
#                    'max_depth': [2, 4, 8]}

max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
n_estimators = [5, 10, 50, 100, 200, 500, 1000]
param_grid = {"max_depth": max_depth, "n_estimators": n_estimators}

# clf=RandomizedSearchCV(XGBClassifier(n_jobs = -1),
#                        tuned_parameters,scoring="accuracy",n_iter=100)
grid = RandomizedSearchCV(
    XGBClassifier(n_jobs=-1), param_grid,
    scoring='neg_log_loss', verbose=2, cv=2
)
grid.fit(final_train, y_train)

print("Best Params:", grid.best_params_)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV] n_estimators=50, max_depth=6 ....................................
[CV] ..................... n_estimators=50, max_depth=6, total=   4.3s
[CV] n_estimators=50, max_depth=6 ....................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    5.0s remaining:    0.0s

[CV] ..................... n_estimators=50, max_depth=6, total=   4.8s
[CV] n_estimators=50, max_depth=8 ....................................
[CV] ..................... n_estimators=50, max_depth=8, total=   4.6s
[CV] n_estimators=50, max_depth=8 ....................................
[CV] ..................... n_estimators=50, max_depth=8, total=   4.8s
[CV] n_estimators=200, max_depth=7 ...................................
[CV] ..................... n_estimators=200, max_depth=7, total=  12.9s
[CV] n_estimators=200, max_depth=7 ...................................
[CV] ..................... n_estimators=200, max_depth=7, total=  13.6s
[CV] n_estimators=10, max_depth=3 ....................................
[CV] ..................... n_estimators=10, max_depth=3, total=   2.1s
[CV] n_estimators=10, max_depth=3 ....................................
[CV] ..................... n_estimators=10, max_depth=3, total=   2.1s
[CV] n_estimators=50, max_depth=10 ...................................
[CV] ..................... n_estimators=50, max_depth=10, total=   4.6s
[CV] n_estimators=50, max_depth=10 ...................................
[CV] ..................... n_estimators=50, max_depth=10, total=   4.8s
[CV] n_estimators=50, max_depth=4 ....................................
[CV] ..................... n_estimators=50, max_depth=4, total=   4.6s
[CV] n_estimators=50, max_depth=4 ....................................
[CV] ..................... n_estimators=50, max_depth=4, total=   5.0s
[CV] n_estimators=5, max_depth=9 .....................................
[CV] ..................... n_estimators=5, max_depth=9, total=   1.8s
[CV] n_estimators=5, max_depth=9 .....................................
[CV] ..................... n_estimators=5, max_depth=9, total=   1.8s
[CV] n_estimators=1000, max_depth=2 ..................................
[CV] ..................... n_estimators=1000, max_depth=2, total=  36.5s
[CV] n_estimators=1000, max_depth=2 ..................................
```

```
[CV] .................. n_estimators=1000, max_depth=2, total=  37.6s
[CV] n_estimators=100, max_depth=7 ....................................
[CV] ................... n_estimators=100, max_depth=7, total=   7.6s
[CV] n_estimators=100, max_depth=7 ....................................
[CV] ................... n_estimators=100, max_depth=7, total=   7.9s
[CV] n_estimators=1000, max_depth=3 ...................................
[CV] .................. n_estimators=1000, max_depth=3, total=  36.2s
[CV] n_estimators=1000, max_depth=3 ...................................
[CV] .................. n_estimators=1000, max_depth=3, total=  38.0s

[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:  4.1min finished

Best Params: {'n_estimators': 1000, 'max_depth': 2}
```

In [76]:
```
clf = XGBClassifier(n_jobs=-1, n_estimators=1000, max_depth = 2)
clf.fit(final_train, y_train)
```

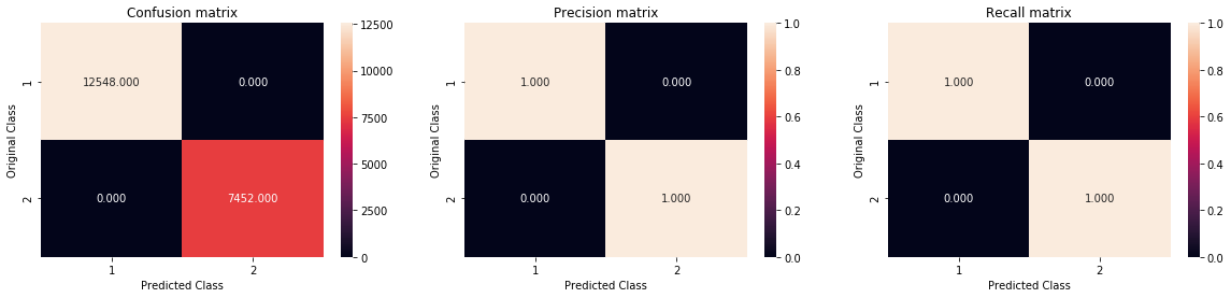
Out[76]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=2, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

In [79]:
```
predict_y = clf.predict_proba(final_train)
print(log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = clf.predict_proba(final_test)
print(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
2.5000924443145323e-05
2.5016929073171924e-05
```

In [81]:
```
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 : 20000
```



Procedure followed

1. Loaded the Question1 and Question2 from given file
2. Splitted the questions and labels in train and test
3. Computed the tfidf of train and test data
4. Horizontally stacked both the question matrices

5. Loaded the already computed basic features and advanced features from database
6. Splitted the features in test and train datasets
7. Converted the features to sparse matrices
8. Horizontally stacked feature matrices and tfidf matrices
9. Trained the tfidf vectors on Logistic Regression
10. Trained the tfidf vectors on Linear SVM
11. Performed RandomizedSearch on XGBoost to find the best parameters from n_estimators and max_depth
12. Trained the XGBoost model on best params got from RandomizedSearch