Quora Question Pair similarity

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
would happen if the Indian government stole the Kohinoor (Koh-i-Noor) di
amond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a
great geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I
see all my Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

3. Exploratory Data Analysis

In [3]:
```
pip install Distance
```

```
Requirement already satisfied: Distance in /usr/local/lib/python3.6/dist-packag
es (0.1.3)
```

In [4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

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

3.1 Reading data and basic stats

In [5]:
```
df = pd.read_csv("/content/drive/My Drive/Quora/train.csv")

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

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

In [6]:
```
df.head()
```

Out[6]:

	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 [7]:
```
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:

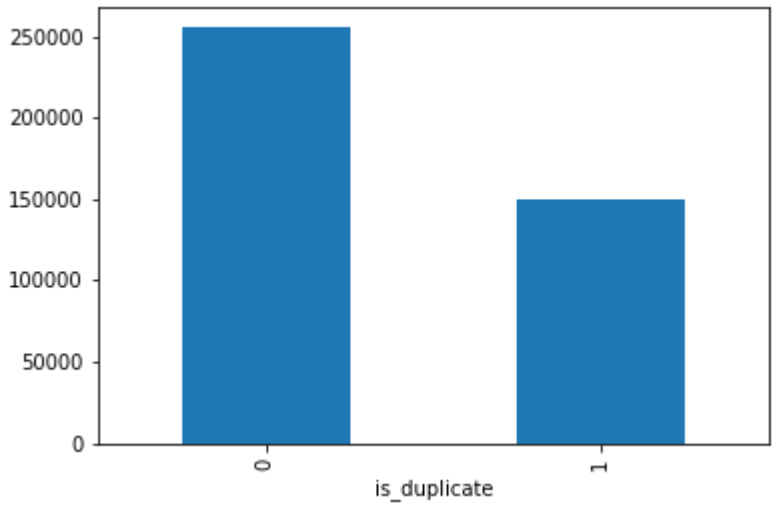
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

In [10]:
```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f390b702e10>
```



In [11]:
```
print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

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

In [12]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round
```

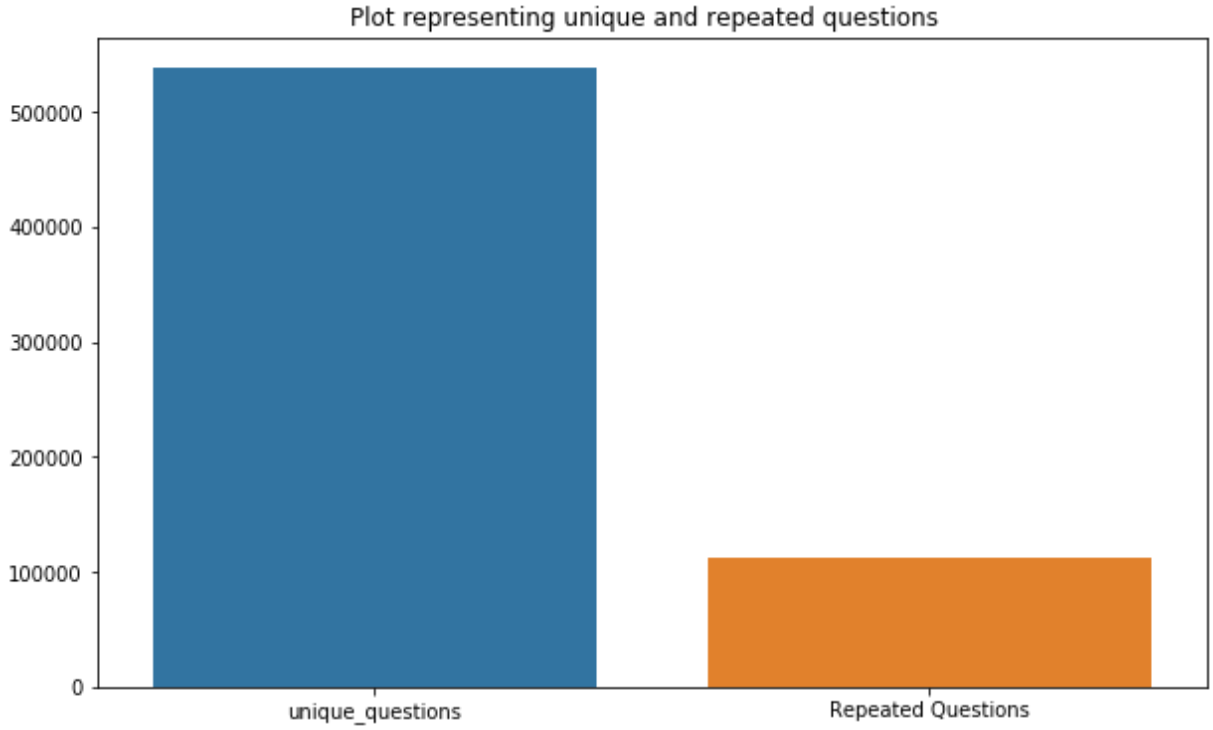
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

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

3.2.2 Number of unique questions

In [13]:
```
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'.f

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 [14]:
```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

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



3.2.3 Checking for Duplicates

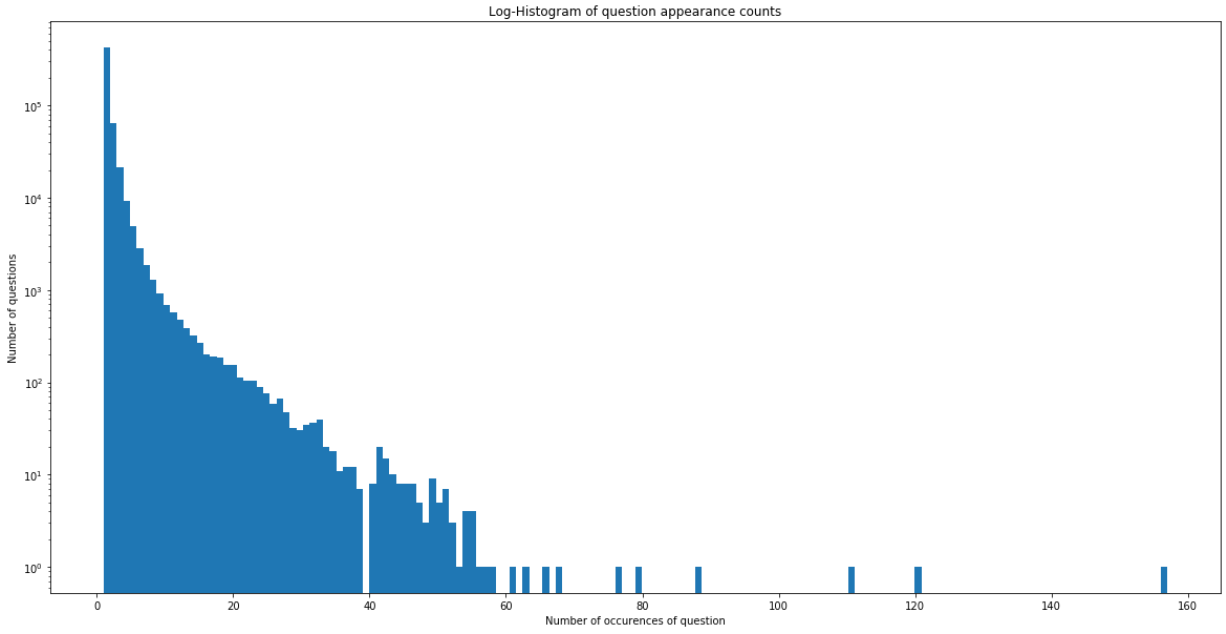
In [15]:
```
#checking whether there are any repeated pair of questions

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 [16]:
```
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(
```

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



3.2.5 Checking for NULL values

In [17]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

```
            id  ...  is_duplicate
105780  105780  ...             0
201841  201841  ...             0
363362  363362  ...             0

[3 rows x 6 columns]
```

- There are two rows with null values in question2

In [18]:
```
# 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 [19]:
```
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[19]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	
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_
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

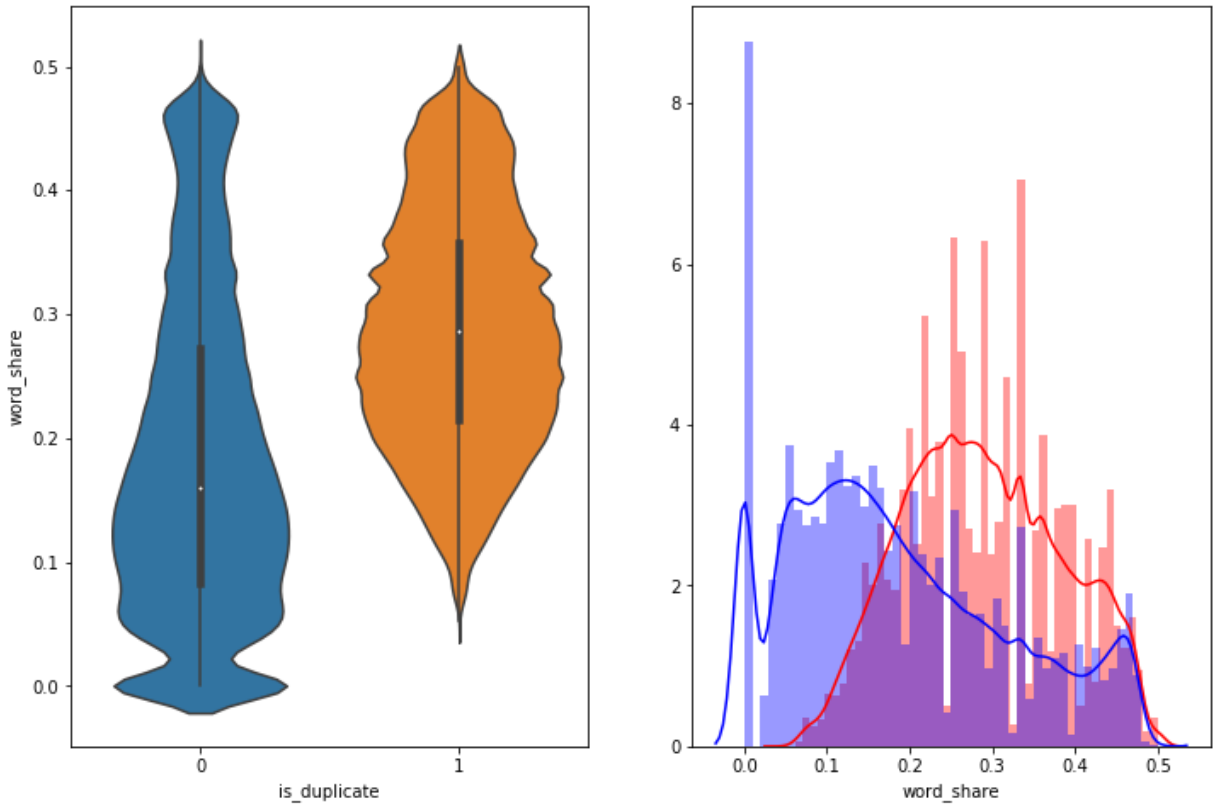
- Here are some questions have only one single words.

In [20]:
```
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 [21]:
```
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", colo
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , col
plt.show()
```

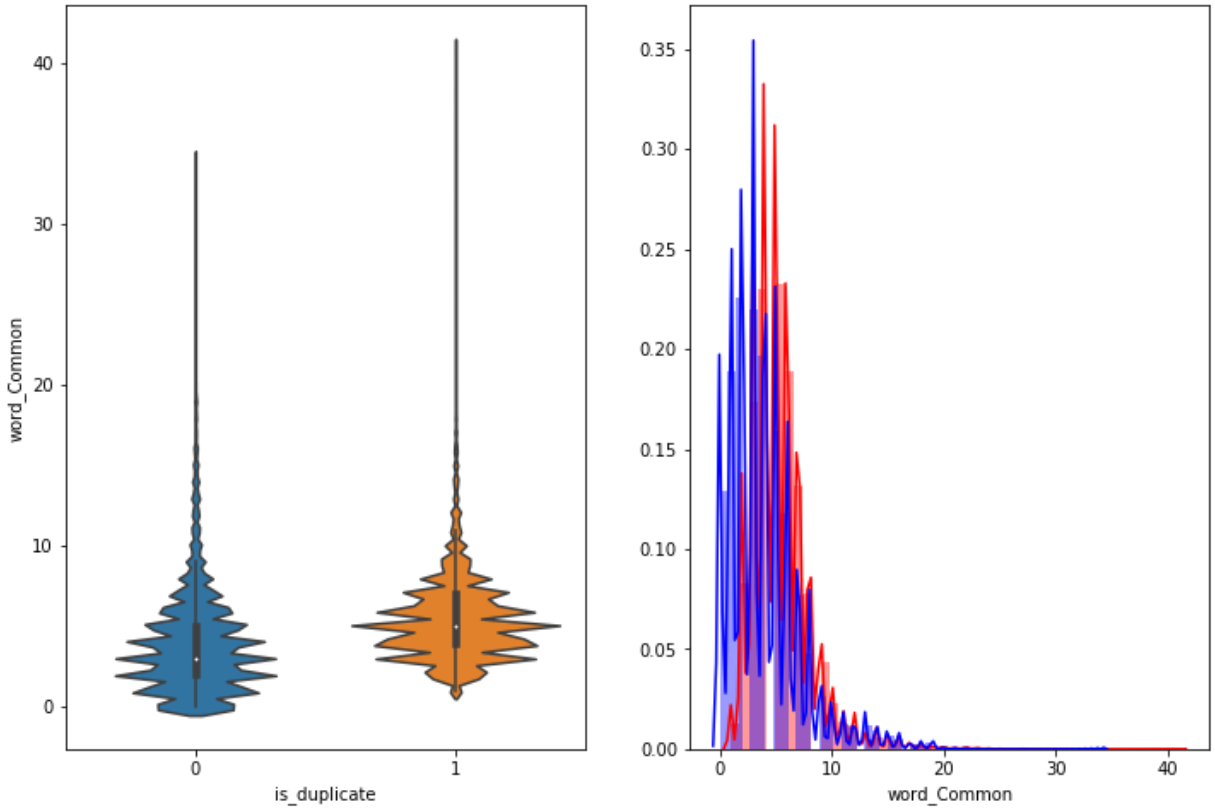


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e.,

there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

In [22]:
```
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" , co
plt.show()
```



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

In [23]:
```
pip install fuzzywuzzy
```

```
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)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

1.2.1 : EDA: Advanced Feature Extraction.

In [8]:
```
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
```

In [0]:
```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previ
```

In [26]:
```
df.head(2)
```

Out[26]:

	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	

3.4 Preprocessing of Text

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

In [27]:
```
import nltk
nltk.download('stopwords')
```

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

Out[27]:
```
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
                           .replace("n't", " not").replace("what's", "what is").
                           .replace("'ve", " have").replace("i'm", "i am").repla
                           .replace("he's", "he is").replace("she's", "she is").
                           .replace("%", " percent ").replace("₹", " rupee ").re
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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


    return x
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-

fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

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

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

In [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["quest
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x[
    return df
```

In [30]:
```
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("/content/drive/My Drive/Quora/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[30]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ct
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	0.9
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	0.6

3.5.1 Analysis of extracted features

3.5.1.1 Plotting Word clouds

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

In [31]:
```
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
```

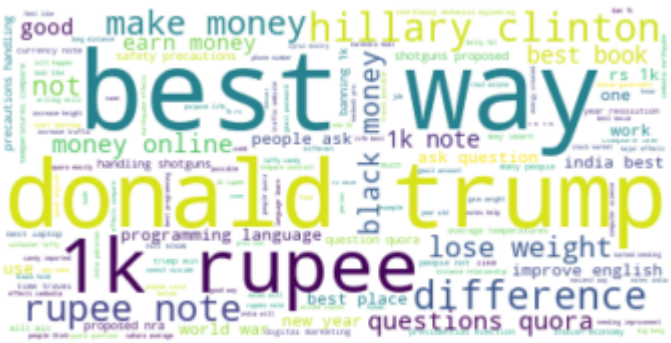
In [32]:
```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

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
```

__ Word Clouds generated from duplicate pair question's text __

In [33]:
```
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
```



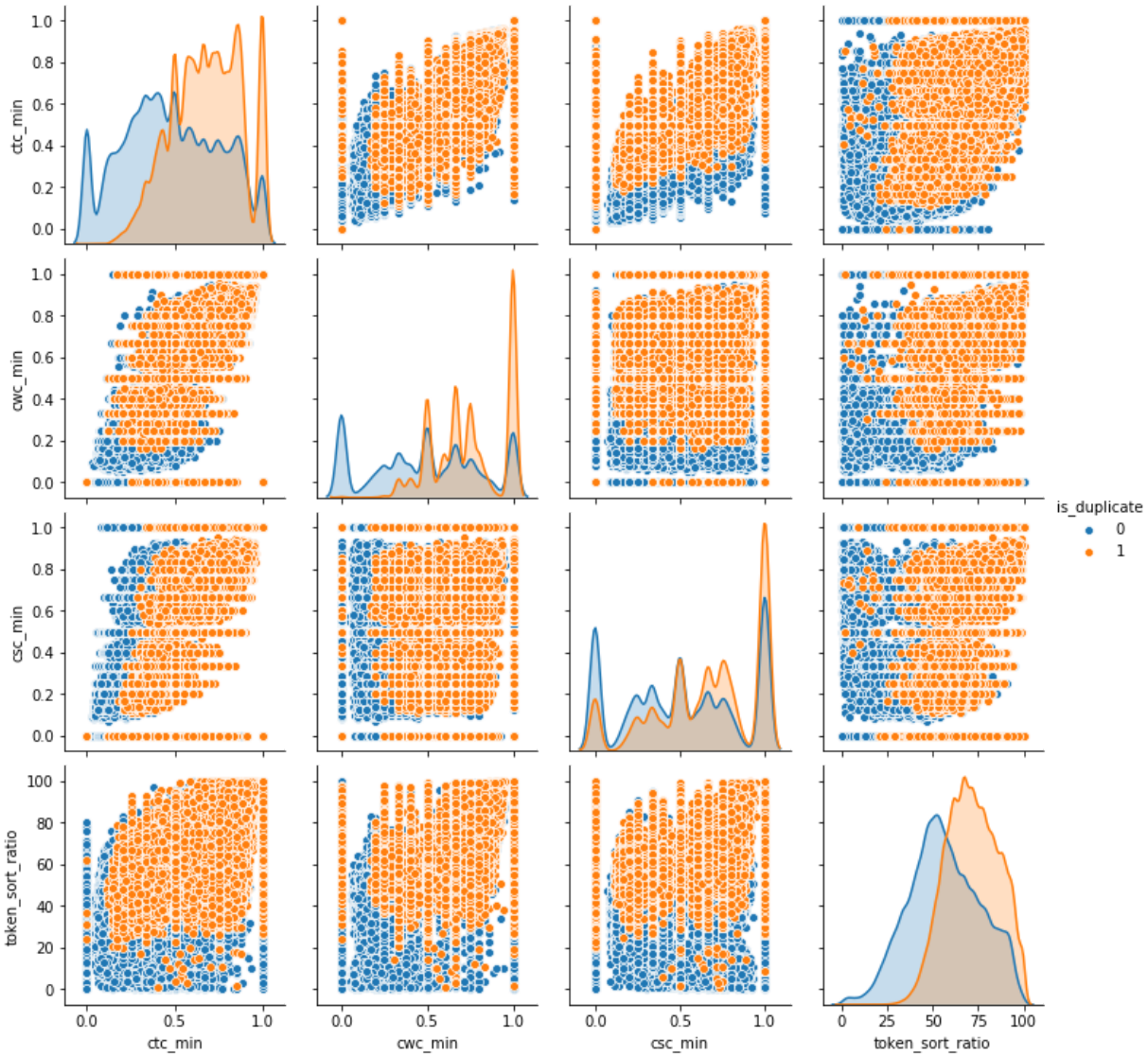
__ Word Clouds generated from non duplicate pair question's text __

In [34]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwo
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

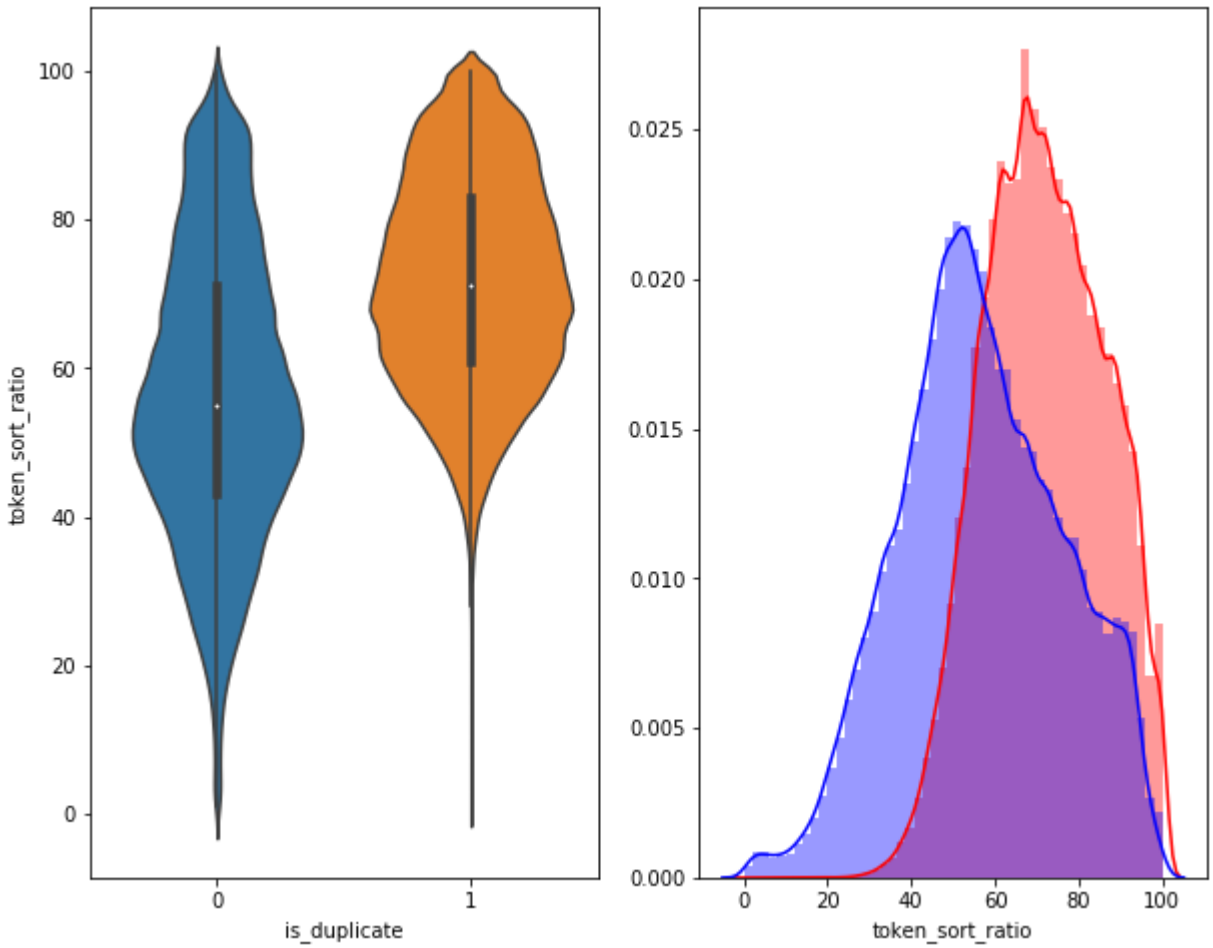
```
Word Cloud for non-Duplicate Question pairs:
```

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

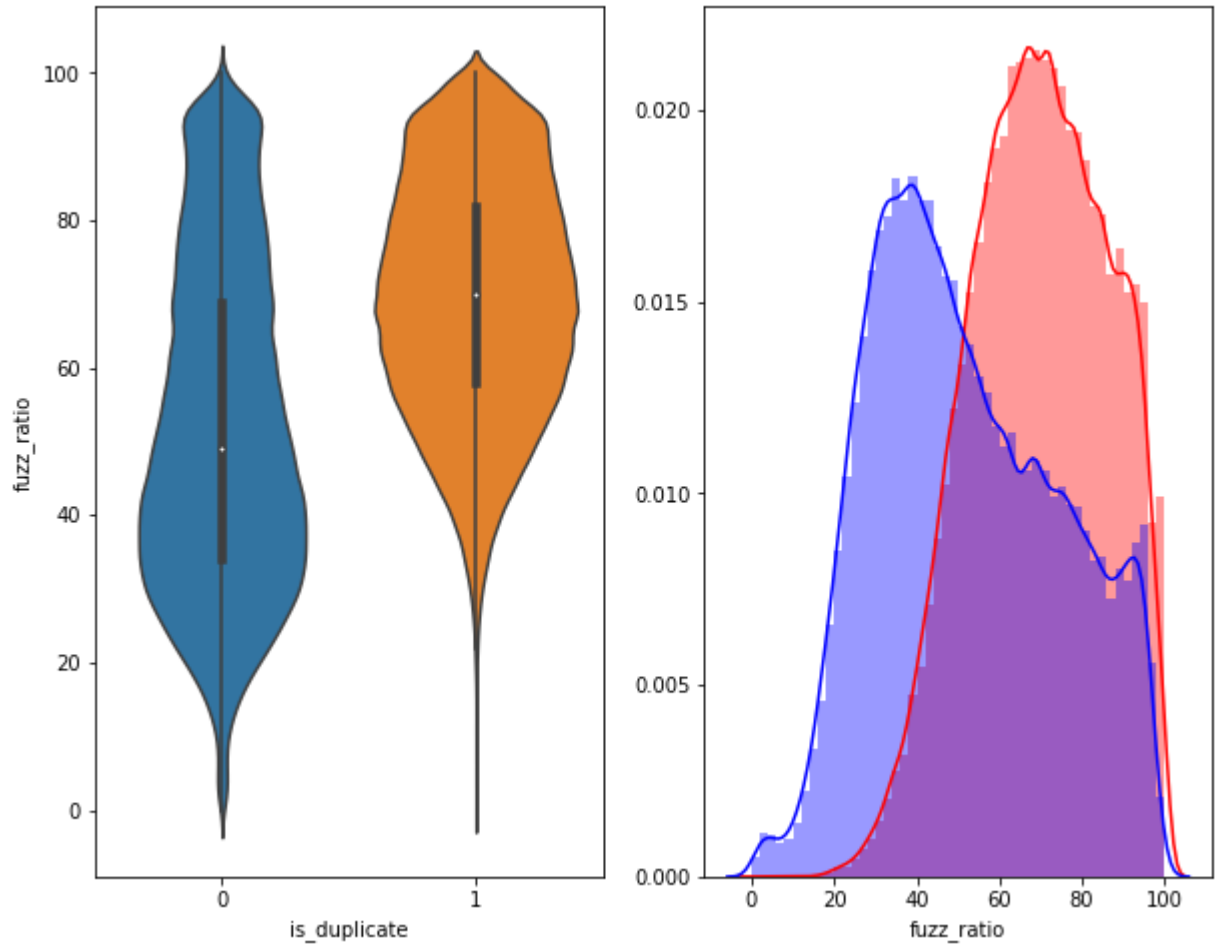
In [35]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplica
plt.show()
```



In [36]:
```
# 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 [37]:
```
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", colo
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , col
plt.show()
```



3.5.2 Visualization

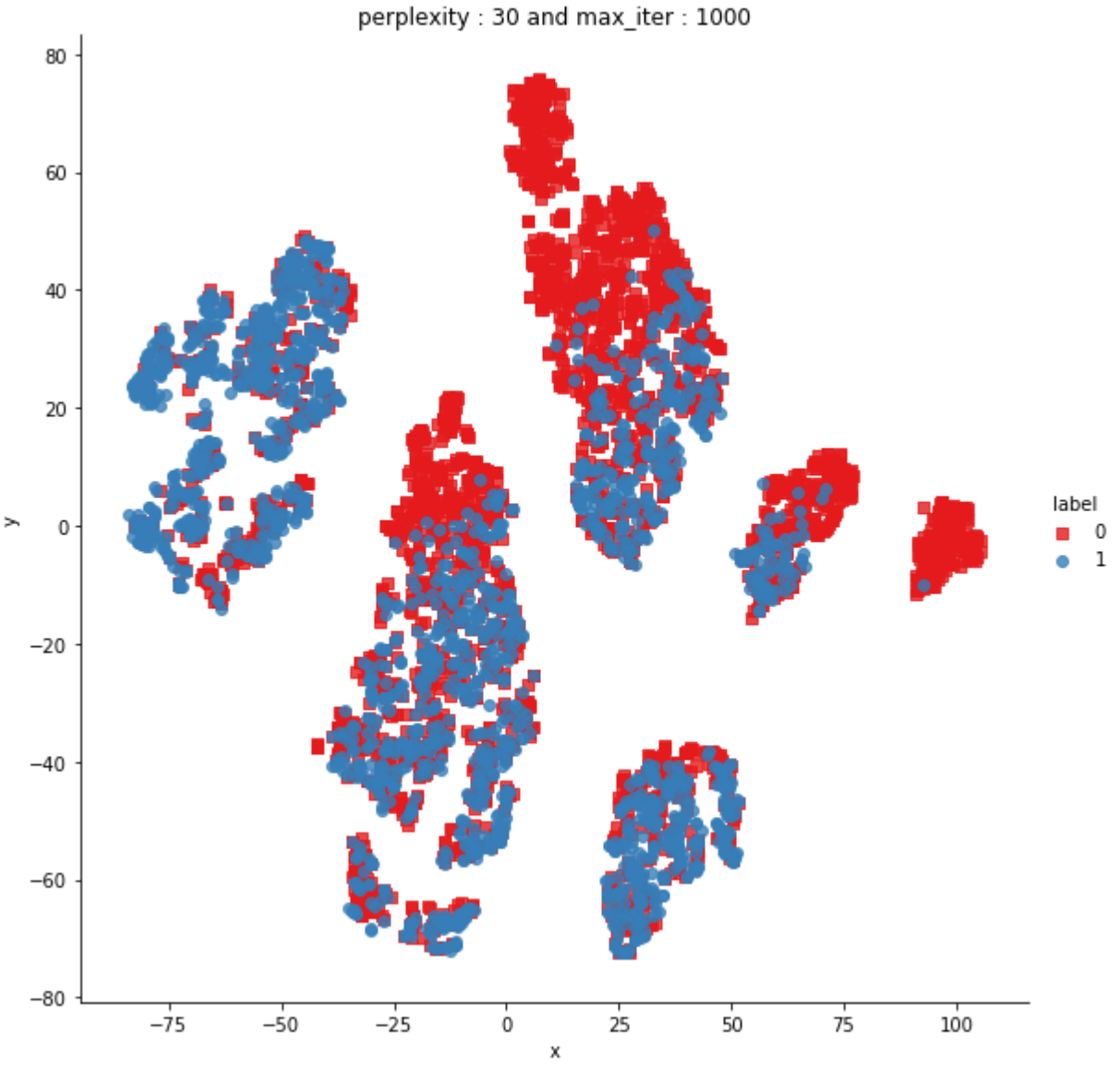
In [0]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleani

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 [0]:
```
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 0.023s...
[t-SNE] Computed neighbors for 5000 samples in 0.345s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.266s
[t-SNE] Iteration 50: error = 81.3425446, gradient norm = 0.0466835 (50 iterati
ons in 1.235s)
[t-SNE] Iteration 100: error = 70.6490860, gradient norm = 0.0087385 (50 iterat
ions in 0.991s)
[t-SNE] Iteration 150: error = 68.9494553, gradient norm = 0.0055224 (50 iterat
ions in 0.893s)
[t-SNE] Iteration 200: error = 68.1286011, gradient norm = 0.0044136 (50 iterat
ions in 0.936s)
[t-SNE] Iteration 250: error = 67.6222382, gradient norm = 0.0040027 (50 iterat
ions in 1.025s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.622238
[t-SNE] Iteration 300: error = 1.7932034, gradient norm = 0.0011886 (50 iterati
ons in 1.066s)
[t-SNE] Iteration 350: error = 1.3933792, gradient norm = 0.0004814 (50 iterati
ons in 0.972s)
[t-SNE] Iteration 400: error = 1.2277225, gradient norm = 0.0002778 (50 iterati
ons in 1.004s)
[t-SNE] Iteration 450: error = 1.1382111, gradient norm = 0.0001874 (50 iterati
ons in 0.940s)
[t-SNE] Iteration 500: error = 1.0834072, gradient norm = 0.0001423 (50 iterati
ons in 0.950s)
[t-SNE] Iteration 550: error = 1.0472494, gradient norm = 0.0001143 (50 iterati
ons in 0.958s)
[t-SNE] Iteration 600: error = 1.0229402, gradient norm = 0.0000992 (50 iterati
ons in 0.966s)
[t-SNE] Iteration 650: error = 1.0064085, gradient norm = 0.0000887 (50 iterati
ons in 0.976s)
[t-SNE] Iteration 700: error = 0.9950163, gradient norm = 0.0000781 (50 iterati
ons in 1.015s)
[t-SNE] Iteration 750: error = 0.9863963, gradient norm = 0.0000739 (50 iterati
ons in 0.981s)
[t-SNE] Iteration 800: error = 0.9797970, gradient norm = 0.0000678 (50 iterati
ons in 0.977s)
[t-SNE] Iteration 850: error = 0.9741811, gradient norm = 0.0000626 (50 iterati
ons in 0.995s)
[t-SNE] Iteration 900: error = 0.9692637, gradient norm = 0.0000620 (50 iterati
ons in 0.990s)
```

```
[t-SNE] Iteration 950: error = 0.9652759, gradient norm = 0.0000559 (50 iterati
ons in 0.997s)
[t-SNE] Iteration 1000: error = 0.9615012, gradient norm = 0.0000559 (50 iterat
ions in 0.974s)
[t-SNE] KL divergence after 1000 iterations: 0.961501
```

In [0]:
```
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="Se
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```




perplexity : 30 and max_iter : 1000
80
60
40
20
0
−20
−40
−60
−80
y
−75
−50
−25
0
25
50
75
100
x
label
0
1

In [0]:
```
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 0.019s...
[t-SNE] Computed neighbors for 5000 samples in 0.434s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.360s
[t-SNE] Iteration 50: error = 80.5739899, gradient norm = 0.0296227 (50 iterati
ons in 4.595s)
[t-SNE] Iteration 100: error = 69.4160385, gradient norm = 0.0032520 (50 iterat
ions in 2.474s)
[t-SNE] Iteration 150: error = 68.0035553, gradient norm = 0.0018662 (50 iterat
ions in 2.105s)
[t-SNE] Iteration 200: error = 67.4419785, gradient norm = 0.0012061 (50 iterat
ions in 2.114s)
[t-SNE] Iteration 250: error = 67.1313705, gradient norm = 0.0008775 (50 iterat
ions in 2.099s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.131371
[t-SNE] Iteration 300: error = 1.5172307, gradient norm = 0.0007258 (50 iterati
ons in 2.766s)
[t-SNE] Iteration 350: error = 1.1812476, gradient norm = 0.0001984 (50 iterati
ons in 3.509s)
[t-SNE] Iteration 400: error = 1.0386292, gradient norm = 0.0000930 (50 iterati
ons in 3.435s)
[t-SNE] Iteration 450: error = 0.9660037, gradient norm = 0.0000607 (50 iterati
ons in 3.463s)
[t-SNE] Iteration 500: error = 0.9280193, gradient norm = 0.0000515 (50 iterati
ons in 3.452s)
[t-SNE] Iteration 550: error = 0.9082615, gradient norm = 0.0000439 (50 iterati
ons in 3.355s)
[t-SNE] Iteration 600: error = 0.8948198, gradient norm = 0.0000341 (50 iterati
ons in 3.374s)
[t-SNE] Iteration 650: error = 0.8839243, gradient norm = 0.0000353 (50 iterati
ons in 3.314s)
[t-SNE] Iteration 700: error = 0.8753766, gradient norm = 0.0000331 (50 iterati
ons in 3.353s)
[t-SNE] Iteration 750: error = 0.8696597, gradient norm = 0.0000279 (50 iterati
ons in 3.549s)
[t-SNE] Iteration 800: error = 0.8648698, gradient norm = 0.0000248 (50 iterati
ons in 3.336s)
[t-SNE] Iteration 850: error = 0.8604140, gradient norm = 0.0000254 (50 iterati
ons in 3.447s)
[t-SNE] Iteration 900: error = 0.8561080, gradient norm = 0.0000236 (50 iterati
```

```
ons in 3.383s)
[t-SNE] Iteration 950: error = 0.8519016, gradient norm = 0.0000246 (50 iterati
ons in 3.371s)
[t-SNE] Iteration 1000: error = 0.8487377, gradient norm = 0.0000225 (50 iterat
ions in 3.429s)
[t-SNE] KL divergence after 1000 iterations: 0.848738
```

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

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

3.6 Featurizing text data with tfidf weighted word-vectors

In [0]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
import spacy
```

In [0]:
```
df_features_1= pd.read_csv("/content/drive/My Drive/Quora/df_fe_without_preproces
```

In [11]:
```
df_features_1.columns
```

Out[11]:
```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

In [0]:
```
df_features_2 = pd.read_csv("/content/drive/My Drive/Quora/nlp_features_train.csv
```

In [13]:
```
df_features_2.columns
```

Out[13]:
```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [0]:
```
df_features_1 = df_features_1.drop(['qid1','qid2'],axis=1)
df_features_2 = df_features_2.drop(['qid1','qid2','question1','question2','is_du
df_features_f = df_features_1.merge(df_features_2, on='id',how='left')
```

In [0]:
```
  df_features_f = df_features_f[df_features_f['question1'].notnull ()]
  df_features_f = df_features_f[df_features_f['question2'].notnull ()]
```

In [18]:
```
df_features_f.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404287 entries, 0 to 404289
Data columns (total 30 columns):
id                      404287 non-null int64
question1               404287 non-null object
question2               404287 non-null object
is_duplicate            404287 non-null int64
freq_qid1               404287 non-null int64
freq_qid2               404287 non-null int64
q1len                   404287 non-null int64
q2len                   404287 non-null int64
q1_n_words              404287 non-null int64
q2_n_words              404287 non-null int64
word_Common             404287 non-null float64
word_Total              404287 non-null float64
word_share              404287 non-null float64
freq_q1+q2              404287 non-null int64
freq_q1-q2              404287 non-null int64
cwc_min                 404287 non-null float64
cwc_max                 404287 non-null float64
csc_min                 404287 non-null float64
csc_max                 404287 non-null float64
ctc_min                 404287 non-null float64
ctc_max                 404287 non-null float64
last_word_eq            404287 non-null float64
first_word_eq           404287 non-null float64
abs_len_diff            404287 non-null float64
mean_len                404287 non-null float64
token_set_ratio         404287 non-null int64
token_sort_ratio        404287 non-null int64
fuzz_ratio              404287 non-null int64
fuzz_partial_ratio      404287 non-null int64
longest_substr_ratio    404287 non-null float64
dtypes: float64(14), int64(14), object(2)
memory usage: 95.6+ MB
```

In [0]:
```
label_1= df_features_f['is_duplicate'][0:100000]
```

In [20]:
```
label_1.shape
```

Out[20]:
```
(100000,)
```

In [0]:
```
df_features_f.drop(['id','is_duplicate'], axis=1,inplace=True)
```

In [0]:
```
 df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
```

In [0]:
```
df=df.iloc[0:100000,:]
```

In [24]:
```
df.shape
```

Out[24]:
```
(100000, 21)
```

In [0]:
```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(df,label_1,stratify=label_1,t
```

In [26]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(70000, 21)
(30000, 21)
(70000,)
(30000,)
```

In [27]:
```
X_train.head()
```

Out[27]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min
29121	29121	877	10437	what are your views on ban of 500 and 1k rupee...	why is the government abruptly banning the 500...	1	0.571420	0.444440	0.399992
57656	57656	101287	101288	are all female porn stars lesbians	are most female pornstars bisexual why	0	0.333322	0.249994	0.499975
83662	83662	141580	141581	have you changed all measuring pparameter and ...	does the uk have higher or lower living standa...	0	0.285710	0.285710	0.399992
79535	79535	135401	135402	what is global citizenship and what are some e...	what is global citizenship what are examples	0	0.999967	0.999967	0.999967
22609	22609	42415	42416	what is your favorite ownhort film	what are your favorite short films	1	0.333322	0.333322	0.666644

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
ques_1_train = list(X_train['question1'].values.astype('U'))
ques_2_train= list(X_train['question2'].values.astype('U'))
ques_1_test = list(X_test['question1'] .values.astype('U'))
ques_2_test= list(X_test['question2'].values.astype('U'))

tfidf_1= TfidfVectorizer(lowercase=False, )
tfidf_1.fit_transform(ques_1_train)
tfidf_1.transform(ques_1_test)

tfidf_2= TfidfVectorizer(lowercase=False, )
tfidf_2.fit_transform(ques_2_train)
tfidf_2.transform(ques_2_test)

# dict key:word and value:tf-idf score
word2tfidf_1 = dict(zip(tfidf_1.get_feature_names(), tfidf_1.idf_))
word2tfidf_2 = dict(zip(tfidf_2.get_feature_names(), tfidf_2.idf_))
```

- After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores.

- here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usage/vectors-similarity (https://spacy.io/usage/vectors-similarity)
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [0]:
```
df1=pd.DataFrame()
df2=pd.DataFrame()
```

vectorizing train data of question 1

In [31]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
ques1=list(X_train['question1'])
for qu1 in tqdm(ques1):
    qu1=str(qu1)
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf_1[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df1['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 70000/70000 [11:05<00:00, 105.21it/s]
```

vectorizing test data of question 1

In [32]:
```
vecs2 = []
ques1=list(X_test['question1'])
for qu1 in tqdm(ques1):
    qu1=str(qu1)
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec2 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec2 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf_1[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df2['q1_feats_m'] = list(vecs2)
```

```
100%|██████████| 30000/30000 [04:43<00:00, 105.89it/s]
```

In [56]:
```
len(vecs1)
```

Out[56]:
```
70000
```

In [53]:
```
len(vec3)
```

Out[53]:
```
96
```

vectorizing train data of question 2

In [33]:
```
vecs3 = []
for qu2 in tqdm(X_train['question2']):
    qu2=str(qu2)
    doc2 = nlp(qu2)
    mean_vec3 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec3 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf_2[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec3 += vec3 * idf
    mean_vec3 = mean_vec3.mean(axis=0)
    vecs3.append(mean_vec3)
df1['q2_feats_m'] = list(vecs3)
```

```
100%|██████████| 70000/70000 [11:03<00:00, 105.47it/s]
```

vectorizing test data of question 2

In [34]:
```
vecs4 = []
for qu2 in tqdm(X_test['question2']):
    qu2=str(qu2)
    doc2 = nlp(qu2)
    mean_vec4 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec4 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf_2[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec4 += vec4 * idf
    mean_vec4 = mean_vec4.mean(axis=0)
    vecs4.append(mean_vec4)
df2['q2_feats_m'] = list(vecs4)
```

```
100%|██████████| 30000/30000 [04:46<00:00, 104.69it/s]
```

In [54]:
```
df1.shape
```

Out[54]:
```
(70000, 2)
```

In [0]:
```
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
```

In [38]:
```
df1.head()
```

Out[38]:

	q1_feats_m	q2_feats_m
0	[181.19857740402222, -21.55388431251049, -59.2...	[80.0660765171051, -48.21563184261322, -73.257...
1	[-7.7056772112846375, -2.099614143371582, -77....	[-0.26102757453918457, -6.536172837018967, -10...
2	[12.566152572631836, -105.02764177322388, -94....	[100.00490683317184, 35.02082224190235, -50.49...
3	[32.555498361587524, -79.1466638147831, -49.22...	[8.723545789718628, -50.16158917546272, -36.14...
4	[41.263129234313965, -59.364133566617966, -34....	[43.181925773620605, -26.223245844244957, -49....

List to DataFrame conversion

In [0]:
```
df3=pd.DataFrame(vecs1)
```

In [0]:
```
df4=pd.DataFrame(vecs2)
```

In [0]:
```
df5=pd.DataFrame(vecs3)
```

In [0]:
```
df6=pd.DataFrame(vecs4)
```

Concatenating Train dataframe of question 1 question 2

In [0]:
```
df7=pd.concat([df3,df5],axis=1)
```

In [77]:
```
df7.shape
```

Out[77]:
```
(70000, 192)
```

Concatenating Test dataframe of question 1 question 2

In [0]:
```
df8=pd.concat([df4,df6],axis=1)
```

In [79]:
```
df8.shape
```

Out[79]:
```
(30000, 192)
```

In [73]:
```
df.head()
```

Out[73]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ct
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	0.9
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	0.6
2	2	5	6	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...	0	0.399992	0.333328	0.399992	0.249997	0.3
3	3	7	8	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...	0	0.000000	0.000000	0.000000	0.000000	0.0
4	4	9	10	which one dissolve in water quikly sugar salt...	which fish would survive in salt water	0	0.399992	0.199998	0.999950	0.666644	0.5

In [95]:
```
df.shape
```

Out[95]:
```
(100000, 21)
```

In [0]:
```
df_train=df.iloc[0:70000,6::]
```

In [97]:
```
df_train.shape
```

Out[97]:
```
(70000, 15)
```

In [0]:
```
df_test=df.iloc[70000:100000,6::]
```

In [94]:
```
df_test.shape
```

Out[94]:
```
(30000, 15)
```

In [0]:
```
df_train=pd.concat([df_train,df7],axis=1)
```

In [99]:
```
print(df_train.shape)
```

```
(70000, 207)
```

In [0]:
```
df_test=pd.concat([df_test,df8],axis=1)
```

In [101]:
```
df_test.shape
```

Out[101]:
```
(60000, 207)
```

4. Machine Learning Models

4.2 Converting strings to numerics

In [0]:
```
df_train= df_train.astype(float)
```

In [0]:
```
df_test=df_test.astype(float)
```

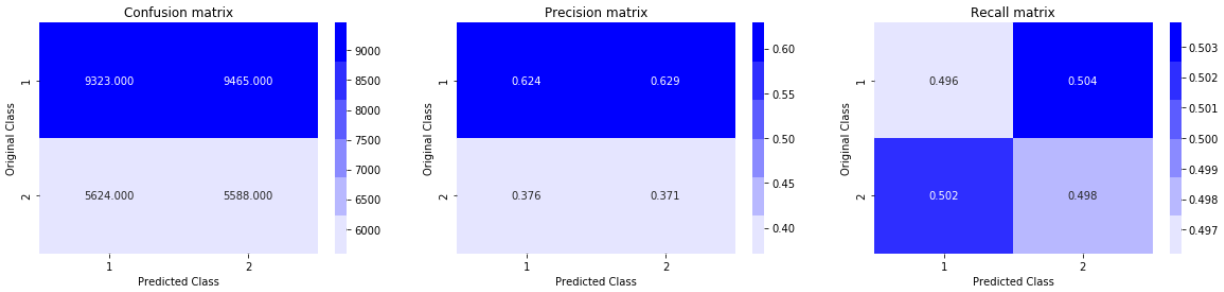
Defining confusion matrix function

In [0]:
```
# 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

    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 rou
    # 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 = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rou
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

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

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

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

    plt.show()
```

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

In [0]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their s
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, e

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

```
Log loss on Test Data using Random Model 0.8886409888419963
```

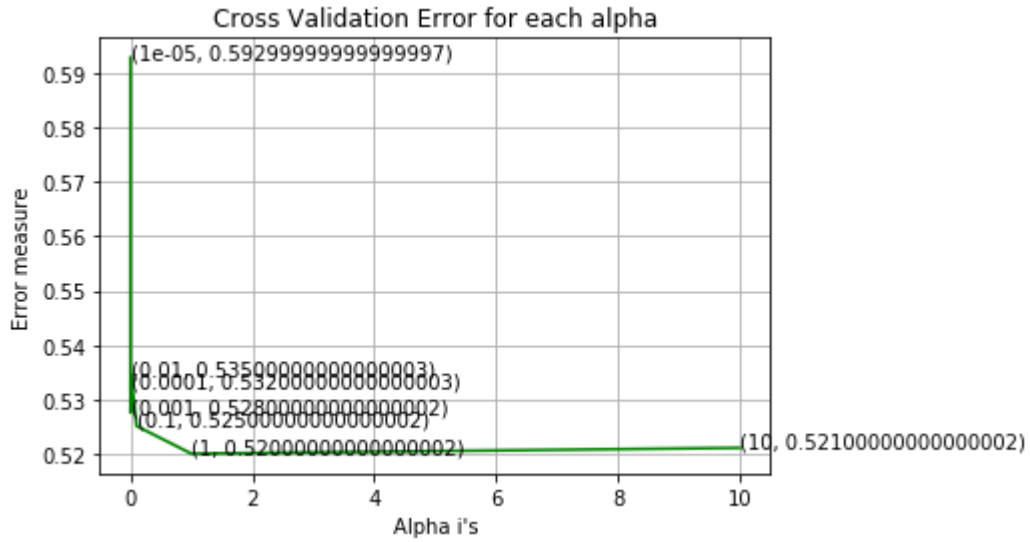


4.4 Logistic Regression with hyperparameter tuning

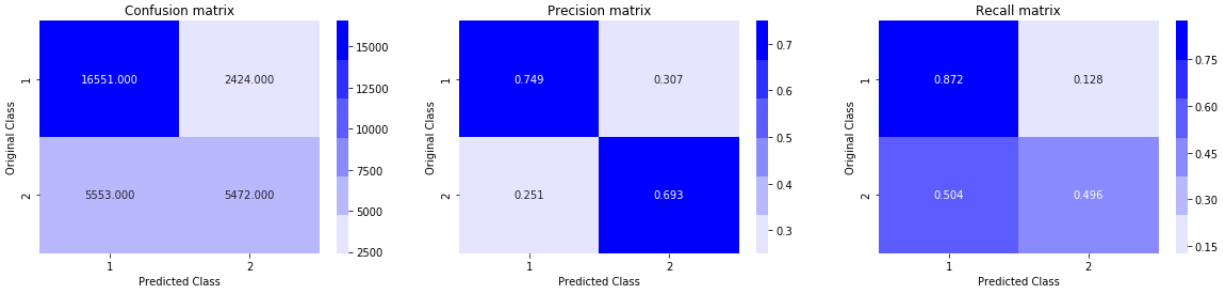
In [0]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Gra
# predict(X)    Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(df_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_train, y_train)
    predict_y = sig_clf.predict_proba(df_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, predi

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


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

predict_y = sig_clf.predict_proba(df_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(df_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
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: 0.592800211149
For values of alpha =  0.0001 The log loss is: 0.532351700629
```

```
For values of alpha =  0.001 The log loss is: 0.527562275995
For values of alpha =  0.01 The log loss is: 0.534535408885
For values of alpha =  0.1 The log loss is: 0.525117052926
For values of alpha =  1 The log loss is: 0.520035530431
For values of alpha =  10 The log loss is: 0.521097925307
```




```
For values of best alpha =  1 The train log loss is: 0.513842874233
For values of best alpha =  1 The test log loss is: 0.520035530431
Total number of data points : 30000
```

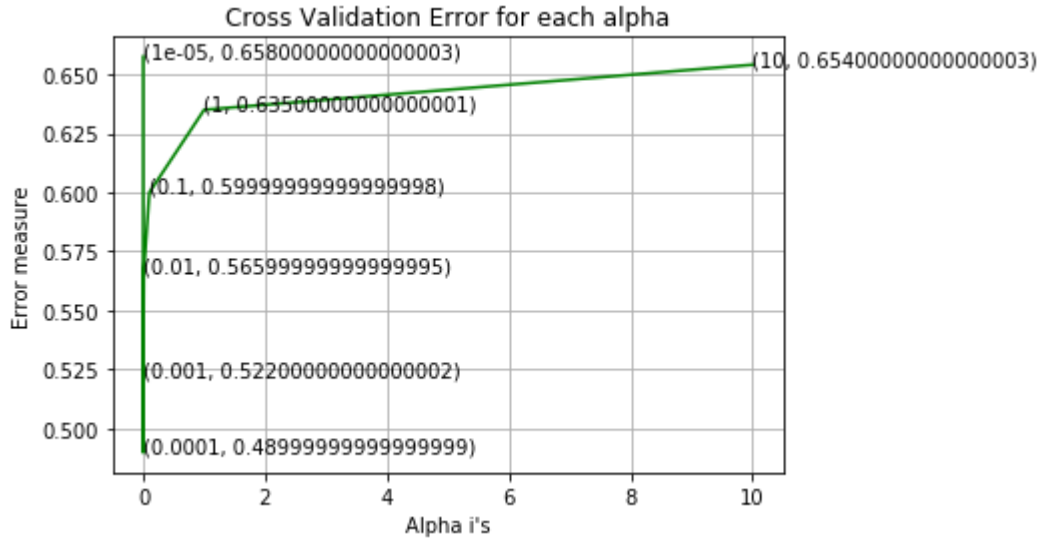


4.5 Linear SVM with hyperparameter tuning

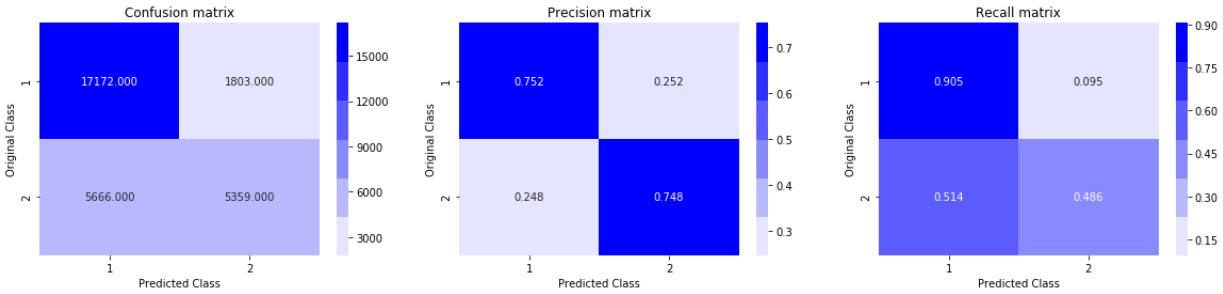
In [0]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Gra
# 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(df_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_train, y_train)
    predict_y = sig_clf.predict_proba(df_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, predi

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_s
clf.fit(df_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(df_train, y_train)

predict_y = sig_clf.predict_proba(df_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(df_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 alpha =  1e-05 The log loss is: 0.657611721261
For values of alpha =  0.0001 The log loss is: 0.489669093534
For values of alpha =  0.001 The log loss is: 0.521829068562
For values of alpha =  0.01 The log loss is: 0.566295616914
```

```
For values of alpha =  0.1 The log loss is: 0.599957866217
For values of alpha =  1 The log loss is: 0.635059427016
For values of alpha =  10 The log loss is: 0.654159467907
```




```
For values of best alpha =  0.0001 The train log loss is: 0.478054677285
For values of best alpha =  0.0001 The test log loss is: 0.489669093534
Total number of data points : 30000
```



4.6 XGBoost

Hyperparameter tuning for XGBoost

In [0]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
params = {'n_estimators' : [100,150,200,400,500] ,'learning_rate' :[0.001, 0.01,
param_grid = params
model_3 =xgb.XGBClassifier(nthread=-1)
r_search = RandomizedSearchCV(model_3, param_grid, scoring="neg_log_loss", n_jobs
r_result = random_search.fit(df_train,y_train)
print("Best: %f using %s" % (r_result.best_score_, r_result.best_params_))
means = r_result.cv_results_['mean_test_score']
stds = r_result.cv_results_['std_test_score']
params = r_result.cv_results_['params']
for mean, stdev, para in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, para))
```

```
Best: -0.347913 using {'n_estimators': 400, 'max_depth': 4, 'learning_rate': 0.
2}
-0.550264 (0.000240) with: {'n_estimators': 400, 'max_depth': 10, 'learning_rat
e': 0.001}
-0.441170 (0.000334) with: {'n_estimators': 500, 'max_depth': 8, 'learning_rat
e': 0.2}
-0.558759 (0.001228) with: {'n_estimators': 400, 'max_depth': 6, 'learning_rat
e': 0.001}
-0.365524 (0.000038) with: {'n_estimators': 200, 'max_depth': 2, 'learning_rat
e': 0.1}
-0.447840 (0.004956) with: {'n_estimators': 200, 'max_depth': 10, 'learning_rat
e': 0.3}
-0.476438 (0.000961) with: {'n_estimators': 100, 'max_depth': 4, 'learning_rat
e': 0.01}
-0.451021 (0.000315) with: {'n_estimators': 100, 'max_depth': 8, 'learning_rat
e': 0.01}
-0.347913 (0.000752) with: {'n_estimators': 400, 'max_depth': 4, 'learning_rat
e': 0.2}
-0.364604 (0.000791) with: {'n_estimators': 200, 'max_depth': 10, 'learning_rat
e': 0.1}
-0.536343 (0.001418) with: {'n_estimators': 500, 'max_depth': 6, 'learning_rat
e': 0.001}
```

In [0]:
```
r_result
```

Out[90]:
```
RandomizedSearchCV(cv=2, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=-1,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'learning_rate': [0.001, 0.01, 0.1, 0.
2,
                                                          0.3],
                                        'max_depth': [2, 4, 6, 8, 10],
                                        'n_estimators': [100, 150, 200, 400,
                                                         500]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='neg_log_loss', verbose=0)
```

Training XGBoost model with best hyper parameters

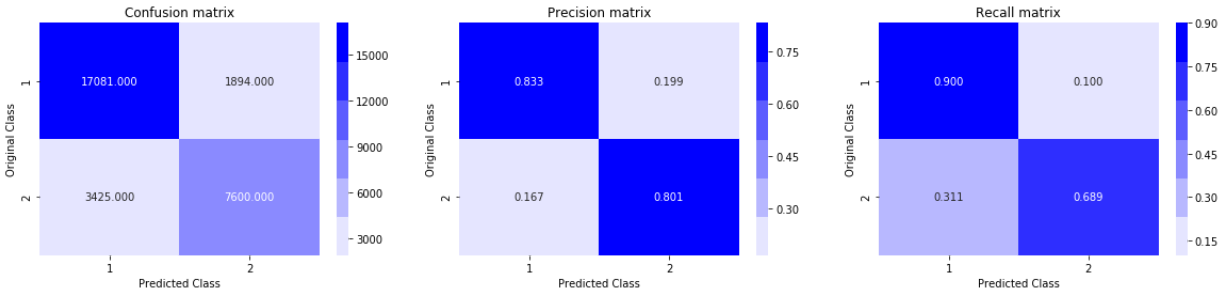
In [0]:
```
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.2
params['max_depth'] = 4
d_train = xgb.DMatrix(df_train, label=y_train)
d_test = xgb.DMatrix(df_test, label=y_test)
watchlist = [(d_train, 'train'), (d_test, 'valid')]
bst = xgb.train(params, d_train,400, watchlist, early_stopping_rounds=20, verbos
xgdmat = xgb.DMatrix(df_train,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, e
```

```
[0]     train-logloss:0.684819  valid-logloss:0.684845
Multiple eval metrics have been passed: 'valid-logloss' will be used for earl
y stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.61583   valid-logloss:0.616104
[20]    train-logloss:0.564616  valid-logloss:0.565273
[30]    train-logloss:0.525758  valid-logloss:0.52679
[40]    train-logloss:0.496661  valid-logloss:0.498021
[50]    train-logloss:0.473563  valid-logloss:0.475182
[60]    train-logloss:0.455315  valid-logloss:0.457186
[70]    train-logloss:0.440442  valid-logloss:0.442482
[80]    train-logloss:0.428424  valid-logloss:0.430795
[90]    train-logloss:0.418803  valid-logloss:0.421447
[100]   train-logloss:0.41069   valid-logloss:0.413583
[110]   train-logloss:0.403831  valid-logloss:0.40693
[120]   train-logloss:0.398076  valid-logloss:0.401402
[130]   train-logloss:0.393305  valid-logloss:0.396851
[140]   train-logloss:0.38913   valid-logloss:0.392952
[150]   train-logloss:0.385469  valid-logloss:0.389521
[160]   train-logloss:0.382327  valid-logloss:0.386667
[170]   train-logloss:0.379541  valid-logloss:0.384148
[180]   train-logloss:0.377014  valid-logloss:0.381932
[190]   train-logloss:0.374687  valid-logloss:0.379883
[200]   train-logloss:0.372585  valid-logloss:0.378068
[210]   train-logloss:0.370615  valid-logloss:0.376367
[220]   train-logloss:0.368559  valid-logloss:0.374595
[230]   train-logloss:0.366545  valid-logloss:0.372847
[240]   train-logloss:0.364708  valid-logloss:0.371311
[250]   train-logloss:0.363021  valid-logloss:0.369886
[260]   train-logloss:0.36144   valid-logloss:0.368673
[270]   train-logloss:0.359899  valid-logloss:0.367421
[280]   train-logloss:0.358465  valid-logloss:0.366395
[290]   train-logloss:0.357128  valid-logloss:0.365361
[300]   train-logloss:0.355716  valid-logloss:0.364315
[310]   train-logloss:0.354425  valid-logloss:0.363403
[320]   train-logloss:0.353276  valid-logloss:0.362595
[330]   train-logloss:0.352084  valid-logloss:0.361823
[340]   train-logloss:0.351051  valid-logloss:0.361167
[350]   train-logloss:0.349867  valid-logloss:0.36043
[360]   train-logloss:0.348829  valid-logloss:0.359773
[370]   train-logloss:0.347689  valid-logloss:0.359019
[380]   train-logloss:0.346607  valid-logloss:0.358311
```

```
[390]   train-logloss:0.345568  valid-logloss:0.357674
The test log loss is: 0.357054433715
```

In [0]:
```
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 : 30000
```



5. Assignments

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Perform hyperparameter tuning of XgBoost models using RandomsearchCV with vectorizer as TF-IDF W2V to reduce the log-loss.

Basic Preprocessing

In [0]:
```
df_features1= pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin
```

In [0]:
```
df_features1.columns
```

Out[67]:
```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words', 'q2_n_words',
       'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

In [0]:
```
df_features2 = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
```

In [0]:
```
df_features2.columns
```

Out[69]:
```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio'],
      dtype='object')
```

In [0]:
```
df_features1 = df_features1.drop(['qid1','qid2'],axis=1)
df_features2 = df_features2.drop(['qid1','qid2','question1','question2','is_dupl
df_features  = df_features1.merge(df_features2, on='id',how='left')
```

In [0]:
```
  df_features = df_features[df_features['question1'].notnull ()]
  df_features = df_features[df_features['question2'].notnull ()]
```

In [0]:
```
df_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404287 entries, 0 to 404289
Data columns (total 30 columns):
id                      404287 non-null int64
question1               404287 non-null object
question2               404287 non-null object
is_duplicate            404287 non-null int64
freq_qid1               404287 non-null int64
freq_qid2               404287 non-null int64
q1len                   404287 non-null int64
q2len                   404287 non-null int64
q1_n_words              404287 non-null int64
q2_n_words              404287 non-null int64
word_Common             404287 non-null float64
word_Total              404287 non-null float64
word_share              404287 non-null float64
freq_q1+q2              404287 non-null int64
freq_q1-q2              404287 non-null int64
cwc_min                 404287 non-null float64
cwc_max                 404287 non-null float64
csc_min                 404287 non-null float64
csc_max                 404287 non-null float64
ctc_min                 404287 non-null float64
ctc_max                 404287 non-null float64
last_word_eq            404287 non-null float64
first_word_eq           404287 non-null float64
abs_len_diff            404287 non-null float64
mean_len                404287 non-null float64
token_set_ratio         404287 non-null int64
token_sort_ratio        404287 non-null int64
fuzz_ratio              404287 non-null int64
fuzz_partial_ratio      404287 non-null int64
longest_substr_ratio    404287 non-null float64
dtypes: float64(14), int64(14), object(2)
memory usage: 95.6+ MB
```

In [0]:
```
label = df_features['is_duplicate']
```

In [0]:
```
df_features.drop(['id','is_duplicate'], axis=1,inplace=True)
```

SPLITTING DATA INTO TRAIN AND TEST

In [0]:
```
X_n_train,X_n_test, y_train, y_test = train_test_split(df_features,label, strat
```

Vectorizing the Questions using TFIDF

In [0]:
```
Vectorizer_1 = TfidfVectorizer()
q1_train = Vectorizer_1.fit_transform(X_n_train['question1'].values. astype('U'))
q1_test= Vectorizer_1.transform(X_n_test['question1'].values. astype('U'))
```

In [0]:
```
Vectorizer_2 = TfidfVectorizer()
q2_train = Vectorizer_2.fit_transform(X_n_train['question2'].values. astype('U'))
q2_test= Vectorizer_2.transform(X_n_test['question2'].values. astype('U'))
```

In [0]:
```
q_train = hstack((q1_train,q2_train))
q_test = hstack((q1_test,q2_test))
```

In [0]:
```
X_n_train.drop(['question1','question2'], axis=1, inplace=True)
X_n_test.drop(['question1','question2'], axis=1, inplace=True)
```

In [0]:
```
df_X_train = hstack((X_n_train, q_train),format="csr" ,dtype='float64')
df_X_test= hstack((X_n_test,q_test),format="csr" ,dtype='float64')
```

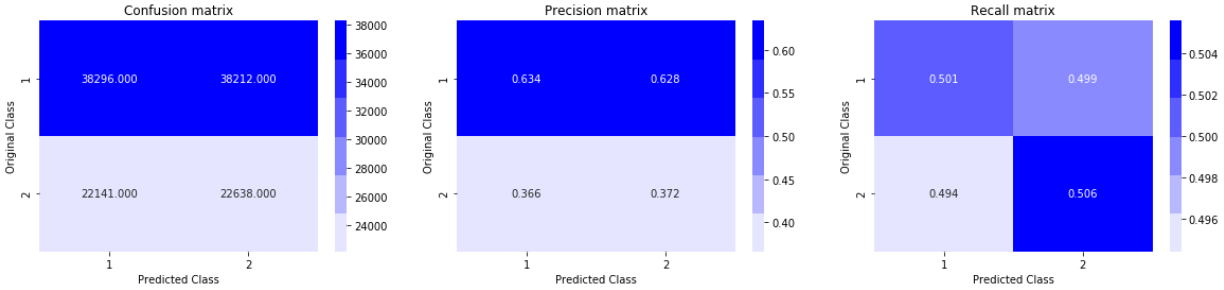
In [0]:
```
print(df_X_train.shape)
print(df_X_test.shape)
```

```
(283000, 113261)
(121287, 113261)
```

Random model to check worst case of a model by its log loss

In [0]:
```
predicted_y = np.zeros((len(y_test),2))
for i in range(len(y_test)):
  rand_probs = np.random.rand(1,2)
  predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, e
predicted_y =np.argmax(predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

```
Log loss on Test Data using Random Model 0.8837482227229594
```



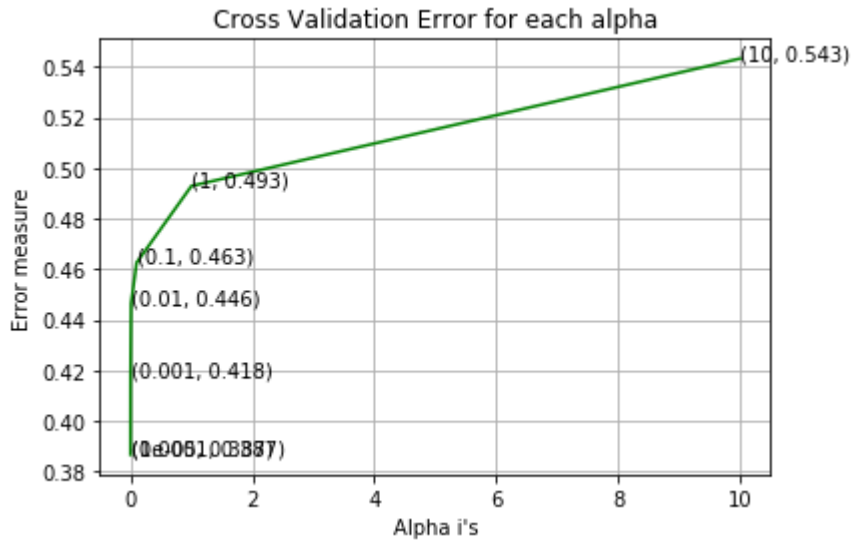
Logistic regression for TFIDF data (400K datapoints)

In [0]:
```
from tqdm import tqdm
```

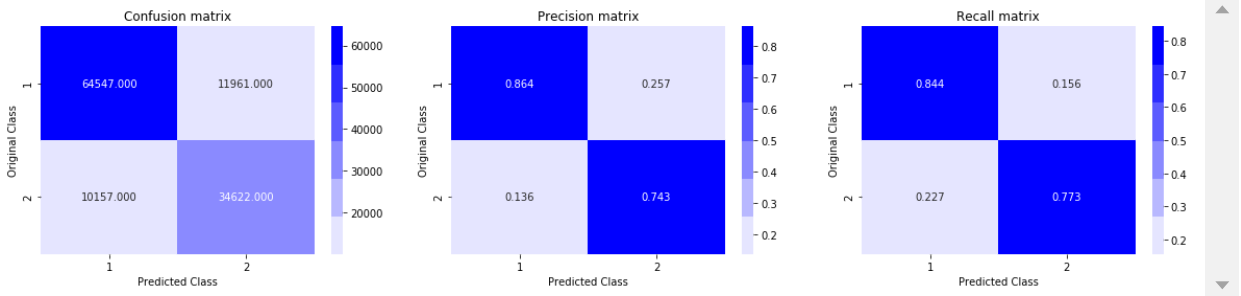
In [0]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Gra
# predict(X)    Predict class labels for samples in X.

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


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

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


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

predict_y = sig_clf.predict_proba(df_X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(df_X_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)
```

```
 14%|█▍        | 1/7 [07:33<45:20, 453.39s/it]
```

```
For values of alpha =  1e-05 The log loss is: 0.3866323867721072
 29%|██▉       | 2/7 [11:52<32:55, 395.03s/it]
For values of alpha =  0.0001 The log loss is: 0.3869806721393733
 43%|████▎     | 3/7 [13:18<20:09, 302.33s/it]
For values of alpha =  0.001 The log loss is: 0.4179480618244989
 57%|█████▋    | 4/7 [13:47<11:01, 220.39s/it]
For values of alpha =  0.01 The log loss is: 0.44643275743608735
 71%|███████   | 5/7 [14:02<05:17, 158.86s/it]
For values of alpha =  0.1 The log loss is: 0.4625394025679981
 86%|████████▌ | 6/7 [14:12<01:53, 114.00s/it]
For values of alpha =  1 The log loss is: 0.49288474499681334
100%|██████████| 7/7 [14:19<00:00, 81.95s/it]
For values of alpha =  10 The log loss is: 0.5432558164400366
```




```
For values of best alpha =  1e-05 The train log loss is: 0.38216247988565416
For values of best alpha =  1e-05 The test log loss is: 0.3866323867721072
Total number of data points : 121287
```

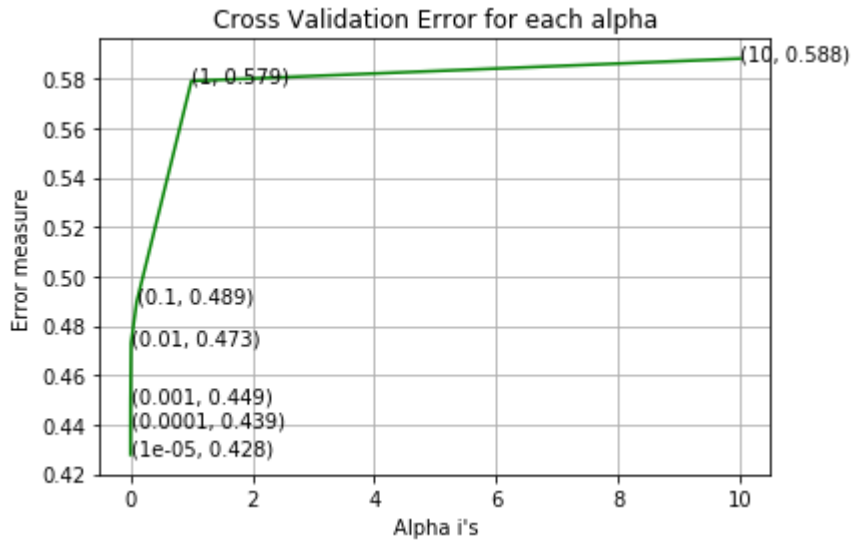


Linear Support Vector Machine for tfidf data (400K data points)

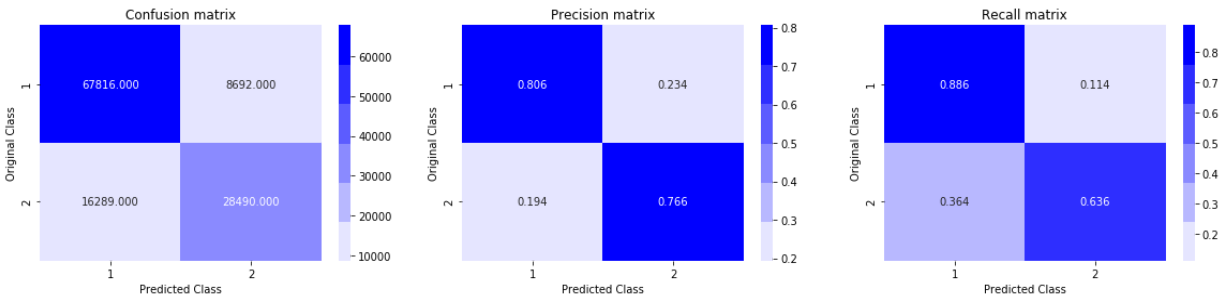
In [0]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Gra
# predict(X)    Predict class labels for samples in X.

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


log_error_array=[]
for i in tqdm(alpha):
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(df_X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_X_train, y_train)
    predict_y = sig_clf.predict_proba(df_X_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, predi

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_s
clf.fit(df_X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(df_X_train, y_train)

predict_y = sig_clf.predict_proba(df_X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(df_X_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)
```

```
 14%|█▍         | 1/7 [06:02<36:12, 362.11s/it]

For values of alpha =  1e-05 The log loss is: 0.42781833956349535
```

```
 29%|██▊         | 2/7 [13:05<31:43, 380.64s/it]
For values of alpha =  0.0001 The log loss is: 0.439329689988059
 43%|████▎       | 3/7 [18:36<24:22, 365.70s/it]
For values of alpha =  0.001 The log loss is: 0.44908361940856495
 57%|█████▋      | 4/7 [22:46<16:32, 330.90s/it]
For values of alpha =  0.01 The log loss is: 0.4729534978805463
 71%|███████▏    | 5/7 [23:57<08:25, 252.95s/it]
For values of alpha =  0.1 The log loss is: 0.48947129790023325
 86%|████████▌   | 6/7 [24:16<03:02, 182.74s/it]
For values of alpha =  1 The log loss is: 0.5789868854192103
100%|██████████| 7/7 [26:28<00:00, 167.41s/it]
For values of alpha =  10 The log loss is: 0.588031125530068
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4246278375323908
For values of best alpha =  1e-05 The test log loss is: 0.42781833956349535
Total number of data points : 121287
```



Summarizing Results in Table form

In [0]:
```
 from prettytable import PrettyTable
p = PrettyTable()
p.field_names = ['Size', 'Model', 'Tokenizer','Train log-loss', 'Test log-loss']
p.add_row(["100K","Random model","TFIDF Weighted W2V","N/A","0.89"])
p.add_row(["100K","Logistic Regression","TFIDF Weighted W2V","0.5138","0.5200"])
p.add_row(["100K","Linear SVM","TFIDF Weighted W2V","0.4780","0.4897"])
p.add_row(["100K","XGBoost","TFIDF Weighted W2V","0.3479","0.3570"])
p.add_row(["400K","Random model","TFIDF","N/A","0.88"])
p.add_row(["400K","Logistic Regression","TFIDF","0.3821","0.3866"])
p.add_row(["400K","Linear SVM","TFIDF","0.4246","0.4278"])
print(p)
```

```
+------+---------------------+--------------------+---------------+-----------
----+
| Size |        Model        |     Tokenizer      | Train log-loss | Test log-l
oss |
+------+---------------------+--------------------+---------------+-----------
----+
| 100K |     Random model    | TFIDF Weighted W2V |      N/A       |      0.89
|
| 100K | Logistic Regression | TFIDF Weighted W2V |     0.5138     |     0.5200
|
| 100K |      Linear SVM     | TFIDF Weighted W2V |     0.4780     |     0.4897
|
| 100K |       XGBoost       | TFIDF Weighted W2V |     0.3479     |     0.3570
|
| 400K |     Random model    |       TFIDF        |      N/A       |      0.88
|
| 400K | Logistic Regression |       TFIDF        |     0.3821     |     0.3866
|
| 400K |      Linear SVM     |       TFIDF        |     0.4246     |     0.4278
|
+------+---------------------+--------------------+---------------+-----------
----+
```

Conclusion:

Logistic regression and Linear SVM work well when we train with higher dimensional data.

XG boost worked well as compared to Logistic regression and Linear SVM with 100K datapoints.

The hyper parameter tuning computational time taken by XG boost is longer as compared to Linear SVM and Logistic regression.So the XG boost may or may not work well at very high dimensional data.