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 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

Columns available in the dataset --> ["id","qid1","qid2","question1","question2","is_duplicate"]

Negative Data Points Examples

1. "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"
2. "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) diamond back?","0"

Positive Data PointsExamples

1. "7","15","16","How can I be a good geologist?","What should I do to be a great geologist?","1"
2. "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

Metric(s):

- log-loss : 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.

Mounting Drive

In [0]:

```
!kill -9 -1
```

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

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

In [1]:

```
!pwd
!ls
```

```
/content
drive  sample_data
```

In [2]:

```
import os
PATH = os.getcwd()
print(PATH)
```

```
/content
```

In [3]:

```
data_path = PATH + '/drive/My Drive/AAIC/Case Studies/Quora Case Study/'
data_path
```

Out[3]:

```
'/content/drive/My Drive/AAIC/Case Studies/Quora Case Study/'
```

3. Exploratory Data Analysis

- *Installing some Modules*

In [0]:

```
pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading
https://files.pythonhosted.org/packages/d8/f1/5a267addb30ab7eaa1beab2b9323073815da4551076554ecc890a
ec9/fuzzywuzzy-0.17.0-py2.py3-none-any.whl
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

In [0]:

```
pip install Distance
```

```
Collecting Distance
  Downloading
https://files.pythonhosted.org/packages/5c/1a/883e47df323437aefa0d0a92ccfb38895d9416bd0b56262c2e46a
7b8/Distance-0.1.3.tar.gz (180kB)
     |████████████████████████████████| 184kB 3.4MB/s
Building wheels for collected packages: Distance
  Building wheel for Distance (setup.py) ... done
  Stored in directory:
/root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d0f12db1c66dbae9c5442b39b001db18e
Successfully built Distance
Installing collected packages: Distance
Successfully installed Distance-0.1.3
```

- *Importing Important modules*

In [4]:

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

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-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

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

3.1 Reading data and basic stats

In [0]:

```
df = pd.read_csv(data_path + "train.csv")

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

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

In [0]:

```
df.head()
```

Out[0]:

	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

- 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

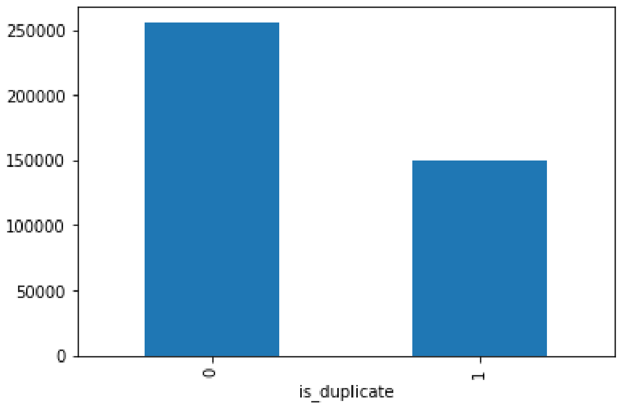
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [0]:

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

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f452f132ef0>
```



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 - 
round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_duplicate'
].mean()*100, 2)))
```

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

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

3.2.2 Number of unique questions

In [0]:

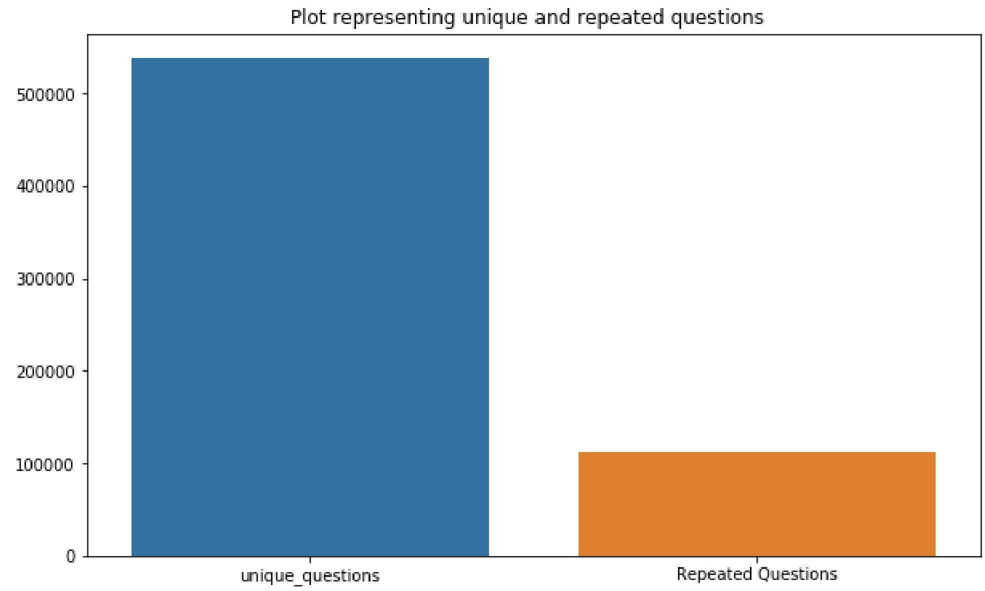
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

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

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

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

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.77953945937505 %)

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()
```



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']).count().reset_index()

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

```
Number of duplicate questions 0
```

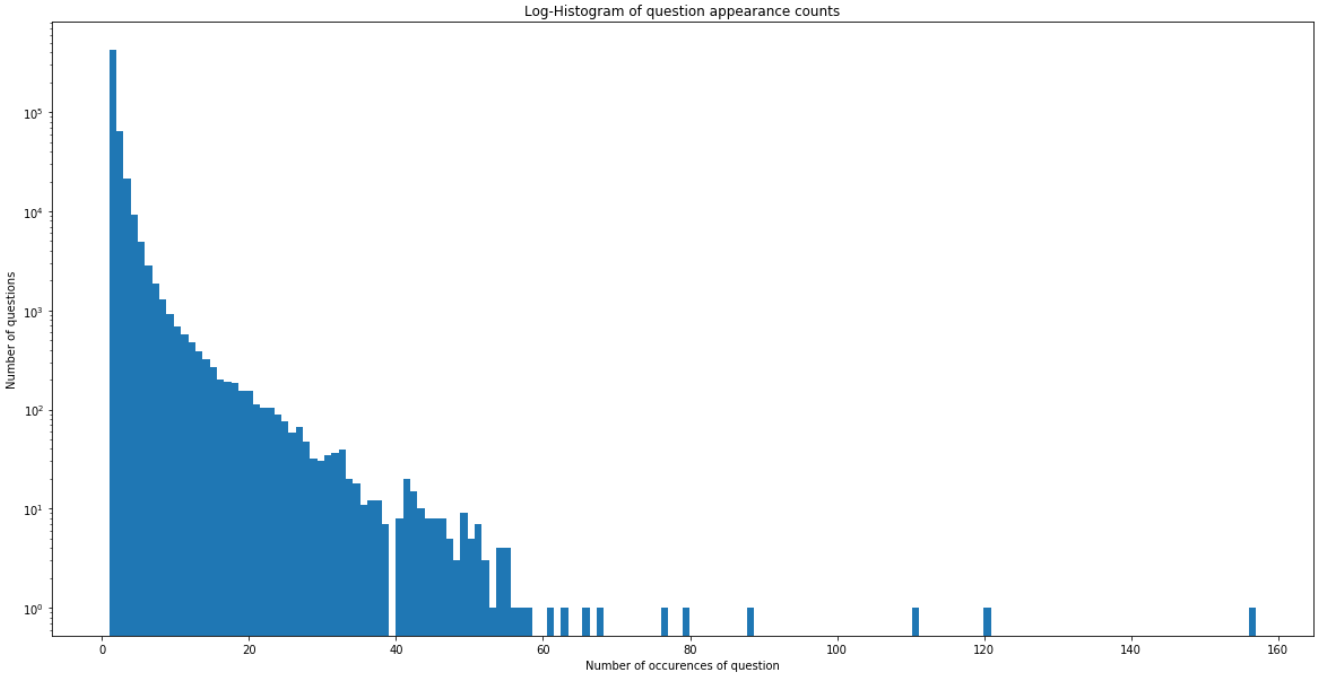
3.2.4 Number of occurrences of each question

In [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(qids.value_counts(
))))
```

```
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  ...  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 [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('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[0]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common
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	14	12	10.0
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	8	13	4.0
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	14	10	4.0

co...

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common
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	11	9	0.0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	13	7	2.0

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [0]:

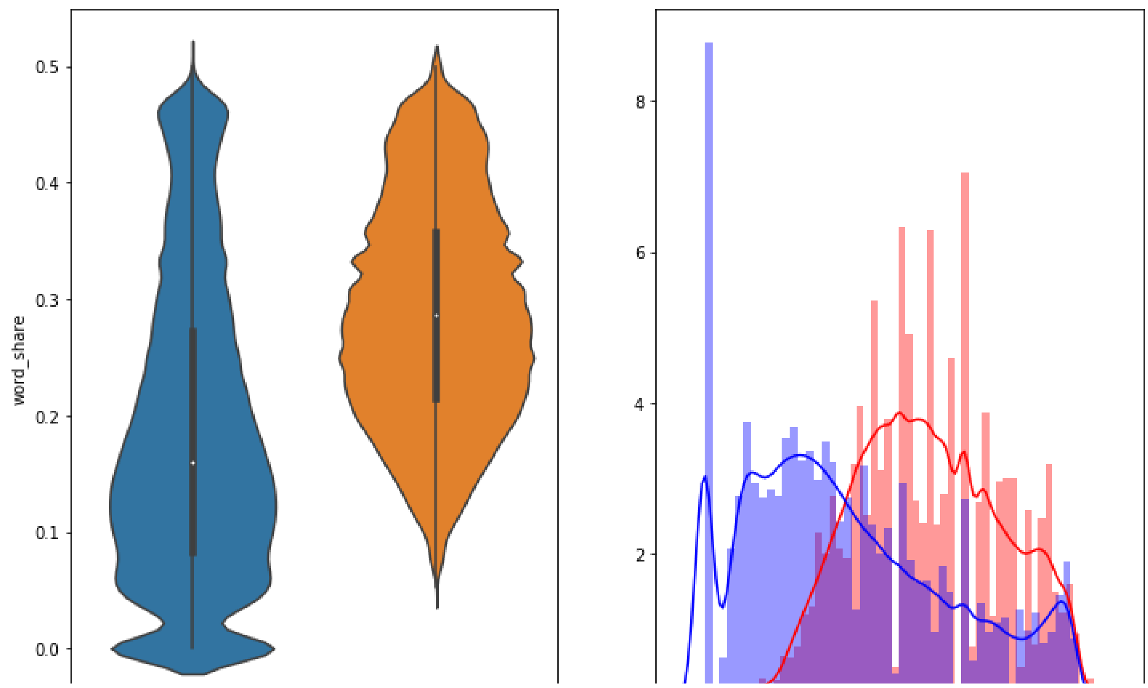
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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

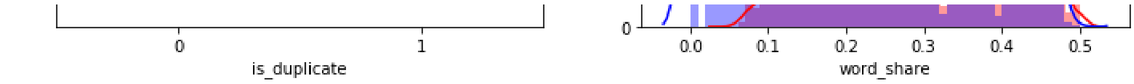
```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

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



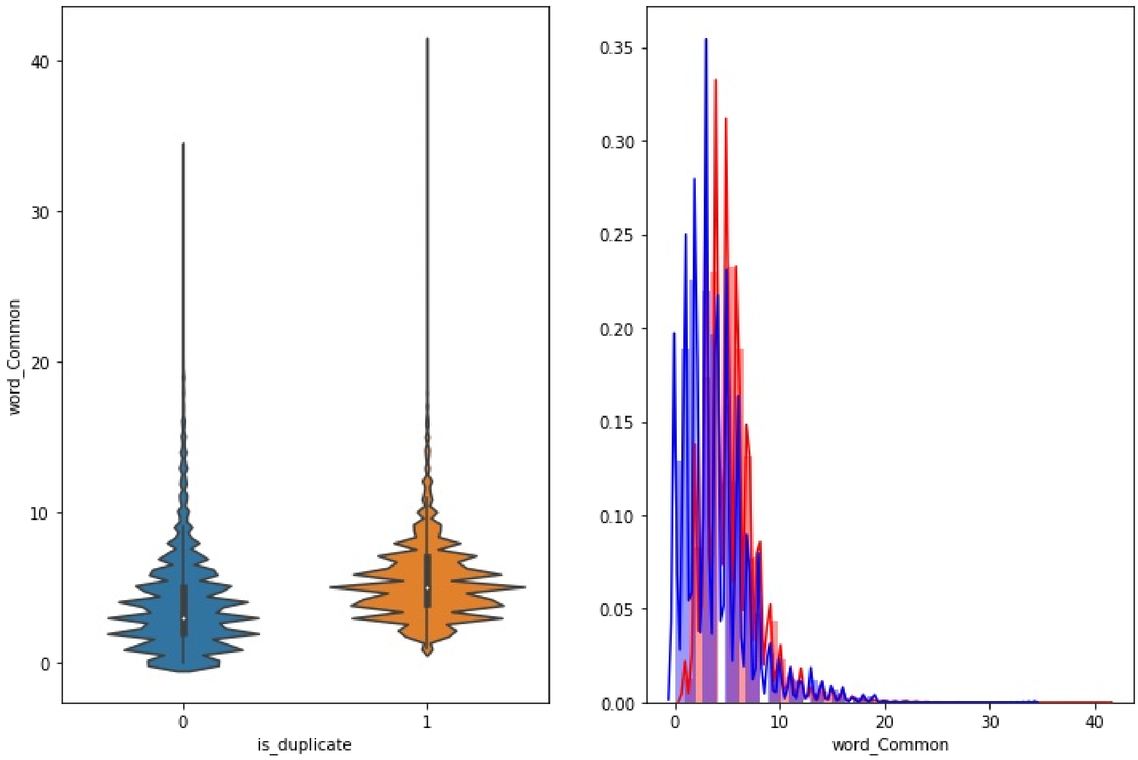


- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

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



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

3.4 Preprocessing of Text

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

In [0]:

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

```
STOP_WORDS = stopwords.words("english")


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


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

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


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


    return x
```

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

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [0]:

```
def get_token_features(q1, q2):
    token_features = [0.0]*10

    # Converting the Sentence into Tokens:
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])

    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])

    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))

    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))

    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)

    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

    # First word of both question is same or not
    token_features[7] = int(q1_tokens[0] == q2_tokens[0])

    token_features[8] = abs(len(q1_tokens) - len(q2_tokens))

    #Average Token Length of both Questions
```

```
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

def extract_features(df):
    # preprocessing each question
    df["question1"] = df["question1"].fillna("").apply(preprocess)
    df["question2"] = df["question2"].fillna("").apply(preprocess)

    print("token features...")

    # Merging Features with dataset

    token_features = df.apply(lambda x: get_token_features(x["question1"], x["question2"]), axis=1)

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

    #Computing Fuzzy Features and Merging with Dataset

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

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["question1"],
x["question2"]), axis=1)
    # The token sort approach involves tokenizing the string in question, sorting the tokens alpha
betically, and
    # then joining them back into a string We then compare the transformed strings with a simple r
atio().
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"],
x["question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), ax
is=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"],
x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(x["question1"], x["qu
estion2"]), axis=1)
    return df
```

In [0]:

```
if os.path.isfile(data_path + 'nlp_features_train.csv'):
    df = pd.read_csv(data_path + "nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv(data_path + "train.csv")
    df = extract_features(df)
    df.to_csv(data_path + "nlp_features_train.csv", index=False)
df.head(2)
```

Out[0]:

id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	firs
			what is the	what is the									

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first
0	0	1	2	step by step guide to invest in sh...	step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	
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	0.699993	0.466664	0.0	

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 [0]:

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

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

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

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

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

In [0]:

```
# 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
```

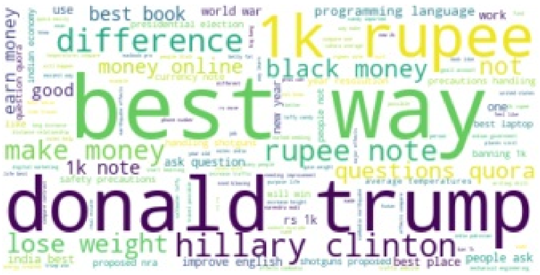
Word Clouds generated from duplicate pair question's text

In [0]:

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

```
plt.axis("off")
plt.show()
```

Word Cloud for Duplicate Question pairs



Word Clouds generated from non duplicate pair question's text

In [0]:

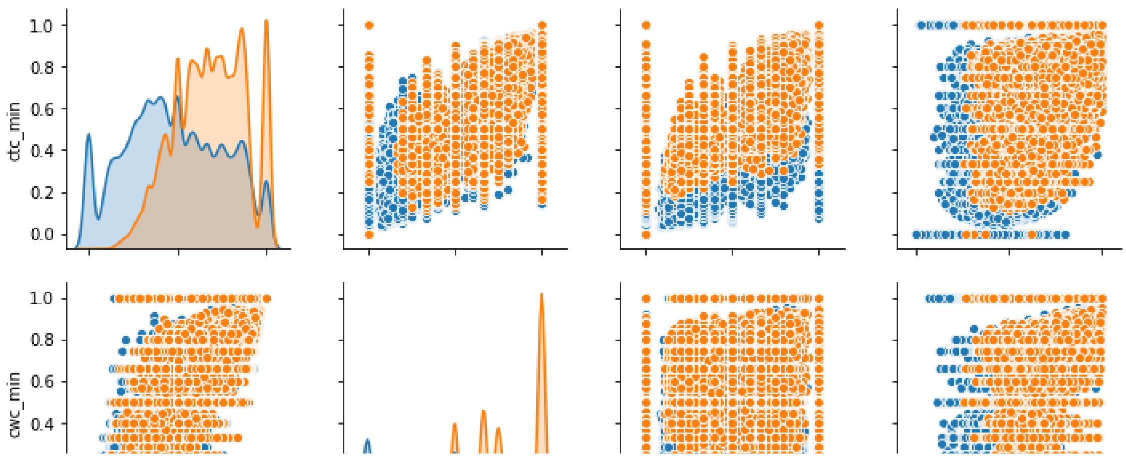
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

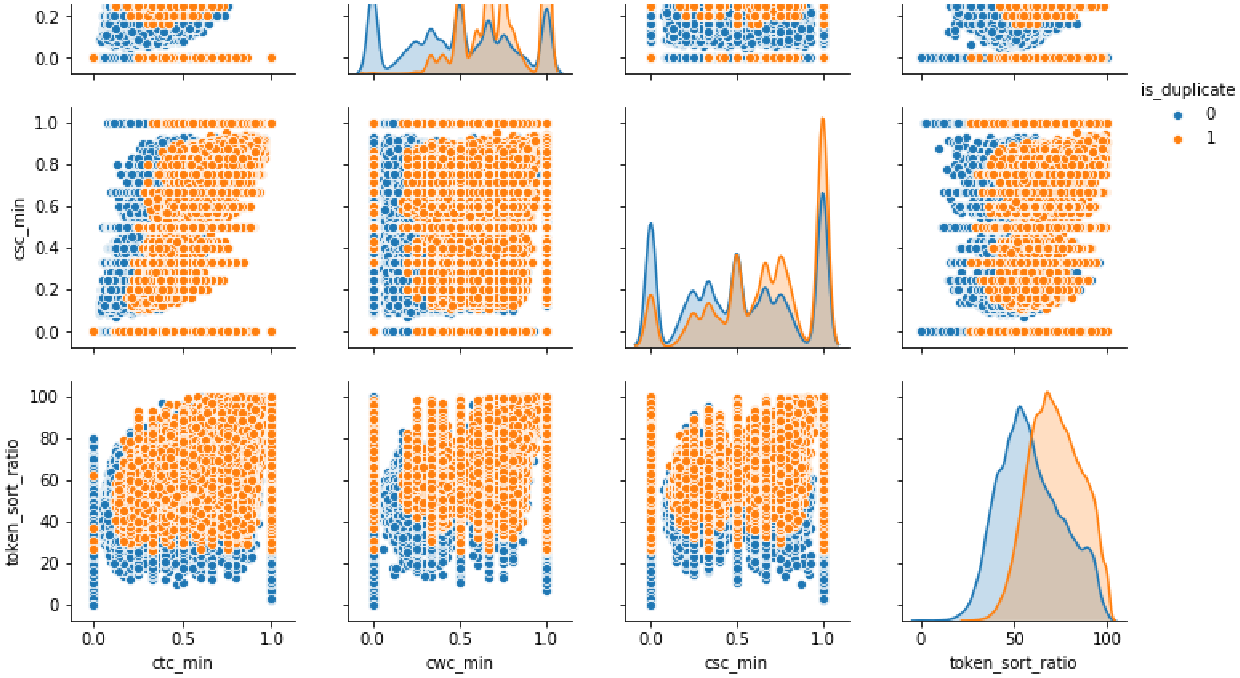
Word Cloud for non-Duplicate Question pairs:

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

In [0]:

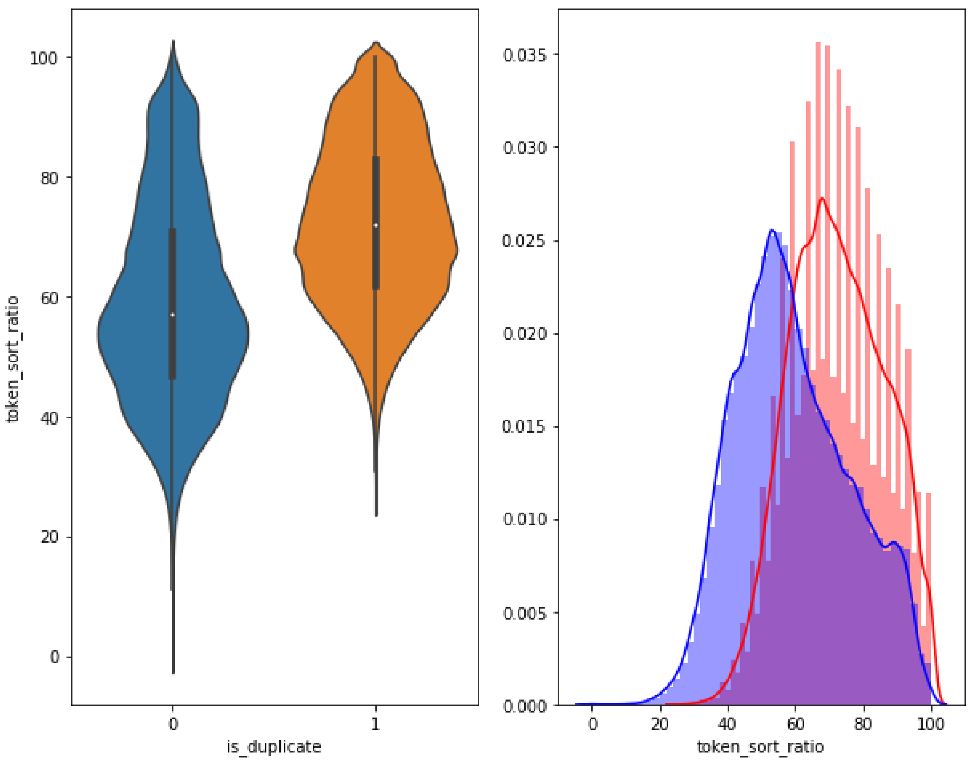
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0:n], hue='i
s_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'])
plt.show()
```





In [0]:

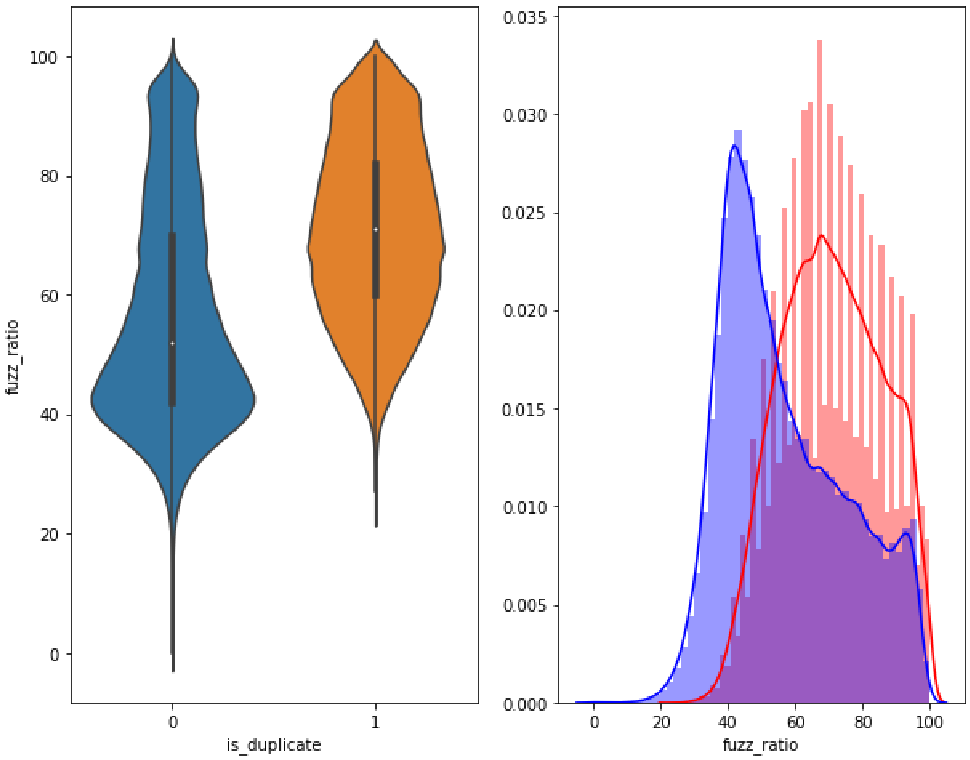
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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

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



In [0]:

```
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 = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , color = 'blue' )
plt.show()
```



3.5.2 Visualization

In [0]:

```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data) to 3
dimention

from sklearn.preprocessing import MinMaxScaler

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

In [0]:

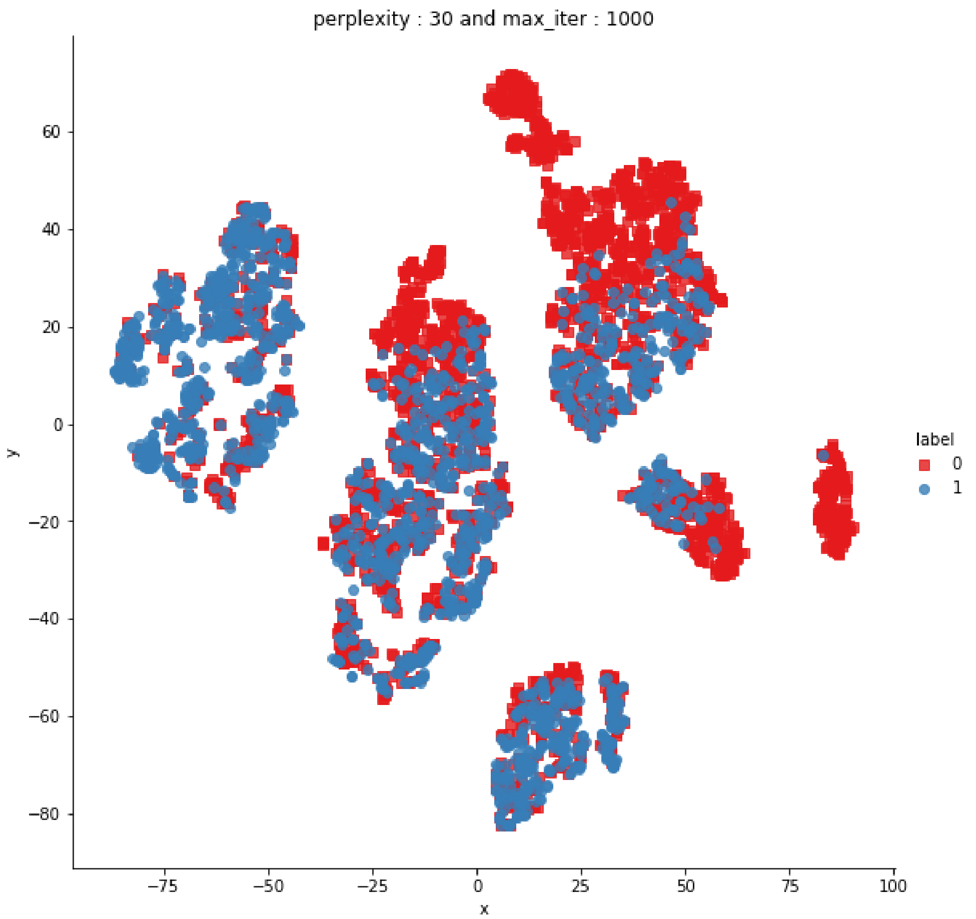
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.013s...
[t-SNE] Computed neighbors for 5000 samples in 0.329s...
[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 0.284s
```

```
[t-SNE] Iteration 50: error = 80.9162369, gradient norm = 0.0427600 (50 iterations in 2.241s)
[t-SNE] Iteration 100: error = 70.3915100, gradient norm = 0.0108003 (50 iterations in 1.683s)
[t-SNE] Iteration 150: error = 68.6126938, gradient norm = 0.0054721 (50 iterations in 1.703s)
[t-SNE] Iteration 200: error = 67.7680206, gradient norm = 0.0042246 (50 iterations in 1.737s)
[t-SNE] Iteration 250: error = 67.2733459, gradient norm = 0.0037275 (50 iterations in 1.794s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.273346
[t-SNE] Iteration 300: error = 1.7734827, gradient norm = 0.0011933 (50 iterations in 1.830s)
[t-SNE] Iteration 350: error = 1.3717980, gradient norm = 0.0004826 (50 iterations in 1.723s)
[t-SNE] Iteration 400: error = 1.2037998, gradient norm = 0.0002772 (50 iterations in 1.744s)
[t-SNE] Iteration 450: error = 1.1133003, gradient norm = 0.0001877 (50 iterations in 1.764s)
[t-SNE] Iteration 500: error = 1.0579894, gradient norm = 0.0001429 (50 iterations in 1.754s)
[t-SNE] Iteration 550: error = 1.0220573, gradient norm = 0.0001178 (50 iterations in 1.755s)
[t-SNE] Iteration 600: error = 0.9990303, gradient norm = 0.0001036 (50 iterations in 1.737s)
[t-SNE] Iteration 650: error = 0.9836842, gradient norm = 0.0000951 (50 iterations in 1.722s)
[t-SNE] Iteration 700: error = 0.9732341, gradient norm = 0.0000860 (50 iterations in 1.735s)
[t-SNE] Iteration 750: error = 0.9649901, gradient norm = 0.0000789 (50 iterations in 1.747s)
[t-SNE] Iteration 800: error = 0.9582695, gradient norm = 0.0000745 (50 iterations in 1.718s)
[t-SNE] Iteration 850: error = 0.9525222, gradient norm = 0.0000732 (50 iterations in 1.726s)
[t-SNE] Iteration 900: error = 0.9479918, gradient norm = 0.0000689 (50 iterations in 1.721s)
[t-SNE] Iteration 950: error = 0.9442031, gradient norm = 0.0000651 (50 iterations in 1.738s)
[t-SNE] Iteration 1000: error = 0.9408465, gradient norm = 0.0000590 (50 iterations in 1.754s)
[t-SNE] KL divergence after 1000 iterations: 0.940847
```

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="Set1",markers=['s','o
'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



3.6 Featurizing text data with tfidf weighted word-vectors

In [0]:

```
import time
```

```python
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy
```

In [0]:

```python
# avoid decoding problems
df = pd.read_csv(data_path + "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 [0]:

```python
df.head()
```

Out[0]:

	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]:

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

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

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

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

In [0]:

```python
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
```

```
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 404290/404290 [53:12<00:00, 126.65it/s]
```

In [0]:

```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 404290/404290 [53:10<00:00, 126.73it/s]
```

In [0]:

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

if os.path.isfile(data_path + 'df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv(data_path + "df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

```
download df_fe_without_preprocessing_train.csv from drive or run previous notebook
```

In [0]:

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

In [0]:

```
# dataframe of nlp features
df1.head()
```

Out[0]:

	id	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
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0

In [0]:

```
# data before preprocessing
df2.head()
```

Out[0]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	freq_q1-q2
0	0	1	1	66	57	14	12	10.0	23.0	0.434783	2	0
1	1	4	1	51	88	8	13	4.0	20.0	0.200000	5	3
2	2	1	1	73	59	14	10	4.0	24.0	0.166667	2	0
3	3	1	1	50	65	11	9	0.0	19.0	0.000000	2	0
4	4	3	1	76	39	13	7	2.0	20.0	0.100000	4	2

In [0]:

```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10
0	211.129864	-144.683059	-68.811247	-153.662141	-89.931593	2.311301	136.743747	50.449112	-64.150964	56.627526	70.148884
1	144.124685	-114.012484	-111.716694	-104.885038	-88.238478	16.441834	58.238013	102.095138	6.026966	178.498497	-22.003573
2	81.757898	-142.184507	0.559867	-104.660084	-84.156631	22.515110	115.521661	50.436953	-111.740923	51.713310	-50.512388
3	-126.651922	-59.747160	-67.763201	-138.114731	-101.038699	88.148523	-22.912261	85.941426	27.784233	50.810650	-64.085183
4	299.444044	-188.632001	-22.946291	-273.683355	-188.480395	107.123044	174.946302	-72.042341	-98.290527	137.439973	-72.358986

5 rows × 96 columns

In [0]:

```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10	
0	151.268526	-127.013168	-31.546286	-142.905807	-97.249094	9.485758	106.682259	36.754201	-36.541905	53.162199	57.798781	34
1	152.023095	-44.955390	-103.559249	-128.467601	-118.567610	44.577916	137.906144	26.984746	-78.328355	86.576880	-38.312126	82
2	4.930220	-29.029581	-117.808812	-98.332275	-19.064096	-9.867805	141.808202	91.269564	50.727205	12.816846	22.755020	61
3	-6.951929	-44.951731	-17.343082	-61.444452	-7.469152	16.942014	95.049250	-2.631600	-13.050916	-28.038393	28.901785	37
4	96.174524	-71.613948	21.584882	-92.742468	-106.643129	10.646790	92.190157	-40.565982	-34.739525	56.340519	-25.369210	36

0	1	2	3	4	5	6	7	8	9	10

5 rows × 96 columns

In [0]:

```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape[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_q1.shape[1]+df3_q2.
shape[1])
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 96
Number of features in question2 w2v  dataframe : 96
Number of features in final dataframe  : 221
```

In [0]:

```
# storing the final features to csv file
if not os.path.isfile(data_path + '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(data_path + 'final_features.csv')
```

4. Machine Learning Models Applying models on TFIDF weighted W2V

4.1 Reading data from file and storing into sql table

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
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
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 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 [0]:

```
#Creating db file from csv
if not os.path.isfile(data_path + 'train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv(data_path + 'final_features.csv', names=['Unnamed: 0','id','is_duplicate'
,'cwc_min','cwc_max','csc_min','csc_max','ctc_min','ctc_max','last_word_eq','first_word_eq','abs_l
en_diff','mean_len','token_set_ratio','token_sort_ratio','fuzz_ratio','fuzz_partial_ratio','longest
_substr_ratio','freq_qid1','freq_qid2','q1len','q2len','q1_n_words','q2_n_words','word_Common','wor
d_Total','word_share','freq_q1+q2','freq_q1-
q2','0_x','1_x','2_x','3_x','4_x','5_x','6_x','7_x','8_x','9_x','10_x','11_x','12_x','13_x','14_x',
'15_x','16_x','17_x','18_x','19_x','20_x','21_x','22_x','23_x','24_x','25_x','26_x','27_x','28_x','
29_x','30_x','31_x','32_x','33_x','34_x','35_x','36_x','37_x','38_x','39_x','40_x','41_x','42_x','4
3_x','44_x','45_x','46_x','47_x','48_x','49_x','50_x','51_x','52_x','53_x','54_x','55_x','56_x','57
_x','58_x','59_x','60_x','61_x','62_x','63_x','64_x','65_x','66_x','67_x','68_x','69_x','70_x','71_
x','72_x','73_x','74_x','75_x','76_x','77_x','78_x','79_x','80_x','81_x','82_x','83_x','84_x','85_x
','86_x','87_x','88_x','89_x','90_x','91_x','92_x','93_x','94_x','95_x','96_x','97_x','98_x','99_x'
,'100_x','101_x','102_x','103_x','104_x','105_x','106_x','107_x','108_x','109_x','110_x','111_x','
112_x','113_x','114_x','115_x','116_x','117_x','118_x','119_x','120_x','121_x','122_x','123_x','12
4_x','125_x','126_x','127_x','128_x','129_x','130_x','131_x','132_x','133_x','134_x','135_x','136_
x','137_x','138_x','139_x','140_x','141_x','142_x','143_x','144_x','145_x','146_x','147_x','148_x'
,'149_x','150_x','151_x','152_x','153_x','154_x','155_x','156_x','157_x','158_x','159_x','160_x','
161_x','162_x','163_x','164_x','165_x','166_x','167_x','168_x','169_x','170_x','171_x','172_x','17
3_x','174_x','175_x','176_x','177_x','178_x','179_x','180_x','181_x','182_x','183_x','184_x','185_
x','186_x','187_x','188_x','189_x','190_x','191_x','192_x','193_x','194_x','195_x','196_x','197_x'
,'198_x','199_x','200_x','201_x','202_x','203_x','204_x','205_x','206_x','207_x','208_x','209_x','
210_x','211_x','212_x','213_x','214_x','215_x','216_x','217_x','218_x','219_x','220_x','221_x','22
2_x','223_x','224_x','225_x','226_x','227_x','228_x','229_x','230_x','231_x','232_x','233_x','234_
x','235_x','236_x','237_x','238_x','239_x','240_x','241_x','242_x','243_x','244_x','245_x','246_x'
,'247_x','248_x','249_x','250_x','251_x','252_x','253_x','254_x','255_x','256_x','257_x','258_x','
259_x','260_x','261_x','262_x','263_x','264_x','265_x','266_x','267_x','268_x','269_x','270_x','27
1_x','272_x','273_x','274_x','275_x','276_x','277_x','278_x','279_x','280_x','281_x','282_x','283_
x','284_x','285_x','286_x','287_x','288_x','289_x','290_x','291_x','292_x','293_x','294_x','295_x'
,'296_x','297_x','298_x','299_x','300_x','301_x','302_x','303_x','304_x','305_x','306_x','307_x','
308_x','309_x','310_x','311_x','312_x','313_x','314_x','315_x','316_x','317_x','318_x','319_x','32
0_x','321_x','322_x','323_x','324_x','325_x','326_x','327_x','328_x','329_x','330_x','331_x','332_
x','333_x','334_x','335_x','336_x','337_x','338_x','339_x','340_x','341_x','342_x','343_x','344_x'
,'345_x','346_x','347_x','348_x','349_x','350_x','351_x','352_x','353_x','354_x','355_x','356_x','
357_x','358_x','359_x','360_x','361_x','362_x','363_x','364_x','365_x','366_x','367_x','368_x','36
9_x','370_x','371_x','372_x','373_x','374_x','375_x','376_x','377_x','378_x','379_x','380_x','381_
x','382_x','383_x','0_y','1_y','2_y','3_y','4_y','5_y','6_y','7_y','8_y','9_y','10_y','11_y','12_y'
,'13_y','14_y','15_y','16_y','17_y','18_y','19_y','20_y','21_y','22_y','23_y','24_y','25_y','26_y',
'27_y','28_y','29_y','30_y','31_y','32_y','33_y','34_y','35_y','36_y','37_y','38_y','39_y','40_y','
41_y','42_y','43_y','44_y','45_y','46_y','47_y','48_y','49_y','50_y','51_y','52_y','53_y','54_y','5
5_y','56_y','57_y','58_y','59_y','60_y','61_y','62_y','63_y','64_y','65_y','66_y','67_y','68_y','69
_y','70_y','71_y','72_y','73_y','74_y','75_y','76_y','77_y','78_y','79_y','80_y','81_y','82_y','83_
y','84_y','85_y','86_y','87_y','88_y','89_y','90_y','91_y','92_y','93_y','94_y','95_y','96_y','97_y
','98_y','99_y','100_y','101_y','102_y','103_y','104_y','105_y','106_y','107_y','108_y','109_y','11
0_y','111_y','112_y','113_y','114_y','115_y','116_y','117_y','118_y','119_y','120_y','121_y','122_
y','123_y','124_y','125_y','126_y','127_y','128_y','129_y','130_y','131_y','132_y','133_y','134_y'
,'135_y','136_y','137_y','138_y','139_y','140_y','141_y','142_y','143_y','144_y','145_y','146_y','
147_y','148_y','149_y','150_y','151_y','152_y','153_y','154_y','155_y','156_y','157_y','158_y','15
9_y','160_y','161_y','162_y','163_y','164_y','165_y','166_y','167_y','168_y','169_y','170_y','171_
y','172_y','173_y','174_y','175_y','176_y','177_y','178_y','179_y','180_y','181_y','182_y','183_y'
,'184_y','185_y','186_y','187_y','188_y','189_y','190_y','191_y','192_y','193_y','194_y','195_y','
196_y','197_y','198_y','199_y','200_y','201_y','202_y','203_y','204_y','205_y','206_y','207_y','20
8_y','209_y','210_y','211_y','212_y','213_y','214_y','215_y','216_y','217_y','218_y','219_y','220_
y','221_y','222_y','223_y','224_y','225_y','226_y','227_y','228_y','229_y','230_y','231_y','232_y'
,'233_y','234_y','235_y','236_y','237_y','238_y','239_y','240_y','241_y','242_y','243_y','244_y','
245_y','246_y','247_y','248_y','249_y','250_y','251_y','252_y','253_y','254_y','255_y','256_y','25
7_y','258_y','259_y','260_y','261_y','262_y','263_y','264_y','265_y','266_y','267_y','268_y','269_
y','270_y','271_y','272_y','273_y','274_y','275_y','276_y','277_y','278_y','279_y','280_y','281_y'
```

```
,'282_y','283_y','284_y','285_y','286_y','287_y','288_y','289_y','290_y','291_y','292_y','293_y','
294_y','295_y','296_y','297_y','298_y','299_y','300_y','301_y','302_y','303_y','304_y','305_y','30
6_y','307_y','308_y','309_y','310_y','311_y','312_y','313_y','314_y','315_y','316_y','317_y','318_
y','319_y','320_y','321_y','322_y','323_y','324_y','325_y','326_y','327_y','328_y','329_y','330_y'
,'331_y','332_y','333_y','334_y','335_y','336_y','337_y','338_y','339_y','340_y','341_y','342_y','
343_y','344_y','345_y','346_y','347_y','348_y','349_y','350_y','351_y','352_y','353_y','354_y','35
5_y','356_y','357_y','358_y','359_y','360_y','361_y','362_y','363_y','364_y','365_y','366_y','367_
y','368_y','369_y','370_y','371_y','372_y','373_y','374_y','375_y','376_y','377_y','378_y','379_y'
,'380_y','381_y','382_y','383_y'], chunksize=chunksize, iterator=True, encoding='utf-8', ):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
```

In [0]:

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

    return None


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

In [0]:

```
import sqlite3
read_db = data_path + 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

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

In [0]:

```
# 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 100001;", conn_r)
        conn_r.commit()
        conn_r.close()
```

In [0]:

```
# 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 [0]:

```
data.head()
```

Out[0]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
1	0.749981250468738	0.374995312558593	0.0	0.0	0.374995312558593	0.230767455634957	(
2	0.66664444518516	0.249996875039062	0.33332222259258	0.111109876556927	0.499991666805553	0.157893905821548	(
3	0.499991666805553	0.199998666675556	0.599988000239995	0.333329629670781	0.545449586821938	0.23999904000384	'
4	0.399992000159997	0.399992000159997	0.499987500312492	0.499987500312492	0.444439506227709	0.39999600004	(
5	0.0	0.0	0.249993750156246	0.166663888935184	0.0555552469152949	0.0416664930562789	(

5 rows × 794 columns

4.2 Converting strings to numerics

In [0]:

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

```
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
q1_n_words
q2_n_words
word_Common
word_Total
word_share
freq_q1+q2
freq_q1-q2
0_x
1_x
2_x
3_x
4_x
5_x
6_x
7_x
8_x
9_x
```

```
10_x
11_x
12_x
13_x
14_x
15_x
16_x
17_x
18_x
19_x
20_x
21_x
22_x
23_x
24_x
25_x
26_x
27_x
28_x
29_x
30_x
31_x
32_x
33_x
34_x
35_x
36_x
37_x
38_x
39_x
40_x
41_x
42_x
43_x
44_x
45_x
46_x
47_x
48_x
49_x
50_x
51_x
52_x
53_x
54_x
55_x
56_x
57_x
58_x
59_x
60_x
61_x
62_x
63_x
64_x
65_x
66_x
67_x
68_x
69_x
70_x
71_x
72_x
73_x
74_x
75_x
76_x
77_x
78_x
79_x
80_x
81_x
82_x
83_x
84_x
85_x
86_x
```

```
87_x
88_x
89_x
90_x
91_x
92_x
93_x
94_x
95_x
96_x
97_x
98_x
99_x
100_x
101_x
102_x
103_x
104_x
105_x
106_x
107_x
108_x
109_x
110_x
111_x
112_x
113_x
114_x
115_x
116_x
117_x
118_x
119_x
120_x
121_x
122_x
123_x
124_x
125_x
126_x
127_x
128_x
129_x
130_x
131_x
132_x
133_x
134_x
135_x
136_x
137_x
138_x
139_x
140_x
141_x
142_x
143_x
144_x
145_x
146_x
147_x
148_x
149_x
150_x
151_x
152_x
153_x
154_x
155_x
156_x
157_x
158_x
159_x
160_x
161_x
162_x
163_x
```

```
164_x
165_x
166_x
167_x
168_x
169_x
170_x
171_x
172_x
173_x
174_x
175_x
176_x
177_x
178_x
179_x
180_x
181_x
182_x
183_x
184_x
185_x
186_x
187_x
188_x
189_x
190_x
191_x
192_x
193_x
194_x
195_x
196_x
197_x
198_x
199_x
200_x
201_x
202_x
203_x
204_x
205_x
206_x
207_x
208_x
209_x
210_x
211_x
212_x
213_x
214_x
215_x
216_x
217_x
218_x
219_x
220_x
221_x
222_x
223_x
224_x
225_x
226_x
227_x
228_x
229_x
230_x
231_x
232_x
233_x
234_x
235_x
236_x
237_x
238_x
239_x
240_x
```

```
241_x
242_x
243_x
244_x
245_x
246_x
247_x
248_x
249_x
250_x
251_x
252_x
253_x
254_x
255_x
256_x
257_x
258_x
259_x
260_x
261_x
262_x
263_x
264_x
265_x
266_x
267_x
268_x
269_x
270_x
271_x
272_x
273_x
274_x
275_x
276_x
277_x
278_x
279_x
280_x
281_x
282_x
283_x
284_x
285_x
286_x
287_x
288_x
289_x
290_x
291_x
292_x
293_x
294_x
295_x
296_x
297_x
298_x
299_x
300_x
301_x
302_x
303_x
304_x
305_x
306_x
307_x
308_x
309_x
310_x
311_x
312_x
313_x
314_x
315_x
316_x
317_x
```

```
318_x
319_x
320_x
321_x
322_x
323_x
324_x
325_x
326_x
327_x
328_x
329_x
330_x
331_x
332_x
333_x
334_x
335_x
336_x
337_x
338_x
339_x
340_x
341_x
342_x
343_x
344_x
345_x
346_x
347_x
348_x
349_x
350_x
351_x
352_x
353_x
354_x
355_x
356_x
357_x
358_x
359_x
360_x
361_x
362_x
363_x
364_x
365_x
366_x
367_x
368_x
369_x
370_x
371_x
372_x
373_x
374_x
375_x
376_x
377_x
378_x
379_x
380_x
381_x
382_x
383_x
0_y
1_y
2_y
3_y
4_y
5_y
6_y
7_y
8_y
9_y
10_y
```

```
11_y
12_y
13_y
14_y
15_y
16_y
17_y
18_y
19_y
20_y
21_y
22_y
23_y
24_y
25_y
26_y
27_y
28_y
29_y
30_y
31_y
32_y
33_y
34_y
35_y
36_y
37_y
38_y
39_y
40_y
41_y
42_y
43_y
44_y
45_y
46_y
47_y
48_y
49_y
50_y
51_y
52_y
53_y
54_y
55_y
56_y
57_y
58_y
59_y
60_y
61_y
62_y
63_y
64_y
65_y
66_y
67_y
68_y
69_y
70_y
71_y
72_y
73_y
74_y
75_y
76_y
77_y
78_y
79_y
80_y
81_y
82_y
83_y
84_y
85_y
86_y
87_y
```

```
88_y
89_y
90_y
91_y
92_y
93_y
94_y
95_y
96_y
97_y
98_y
99_y
100_y
101_y
102_y
103_y
104_y
105_y
106_y
107_y
108_y
109_y
110_y
111_y
112_y
113_y
114_y
115_y
116_y
117_y
118_y
119_y
120_y
121_y
122_y
123_y
124_y
125_y
126_y
127_y
128_y
129_y
130_y
131_y
132_y
133_y
134_y
135_y
136_y
137_y
138_y
139_y
140_y
141_y
142_y
143_y
144_y
145_y
146_y
147_y
148_y
149_y
150_y
151_y
152_y
153_y
154_y
155_y
156_y
157_y
158_y
159_y
160_y
161_y
162_y
163_y
164_y
```

```
165_y
166_y
167_y
168_y
169_y
170_y
171_y
172_y
173_y
174_y
175_y
176_y
177_y
178_y
179_y
180_y
181_y
182_y
183_y
184_y
185_y
186_y
187_y
188_y
189_y
190_y
191_y
192_y
193_y
194_y
195_y
196_y
197_y
198_y
199_y
200_y
201_y
202_y
203_y
204_y
205_y
206_y
207_y
208_y
209_y
210_y
211_y
212_y
213_y
214_y
215_y
216_y
217_y
218_y
219_y
220_y
221_y
222_y
223_y
224_y
225_y
226_y
227_y
228_y
229_y
230_y
231_y
232_y
233_y
234_y
235_y
236_y
237_y
238_y
239_y
240_y
241_y
```

```
241_y
242_y
243_y
244_y
245_y
246_y
247_y
248_y
249_y
250_y
251_y
252_y
253_y
254_y
255_y
256_y
257_y
258_y
259_y
260_y
261_y
262_y
263_y
264_y
265_y
266_y
267_y
268_y
269_y
270_y
271_y
272_y
273_y
274_y
275_y
276_y
277_y
278_y
279_y
280_y
281_y
282_y
283_y
284_y
285_y
286_y
287_y
288_y
289_y
290_y
291_y
292_y
293_y
294_y
295_y
296_y
297_y
298_y
299_y
300_y
301_y
302_y
303_y
304_y
305_y
306_y
307_y
308_y
309_y
310_y
311_y
312_y
313_y
314_y
315_y
316_y
317_y
318_y
```

```
318_y
319_y
320_y
321_y
322_y
323_y
324_y
325_y
326_y
327_y
328_y
329_y
330_y
331_y
332_y
333_y
334_y
335_y
336_y
337_y
338_y
339_y
340_y
341_y
342_y
343_y
344_y
345_y
346_y
347_y
348_y
349_y
350_y
351_y
352_y
353_y
354_y
355_y
356_y
357_y
358_y
359_y
360_y
361_y
362_y
363_y
364_y
365_y
366_y
367_y
368_y
369_y
370_y
371_y
372_y
373_y
374_y
375_y
376_y
377_y
378_y
379_y
380_y
381_y
382_y
383_y
```

In [0]:

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

4.3 Random train test split(70:30)

In [0]:

```
X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

In [0]:

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

```
Number of data points in train data : (70000, 794)
Number of data points in test data : (30000, 794)
```

In [0]:

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

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6302428571428571 Class 1:  0.36975714285714284
---------- Distribution of output variable in train data ----------
Class 0:  0.36973333333333336 Class 1:  0.36973333333333336
```

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 predicted class j

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

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

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

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #      [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

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

    plt.subplot(1, 3, 2)
```

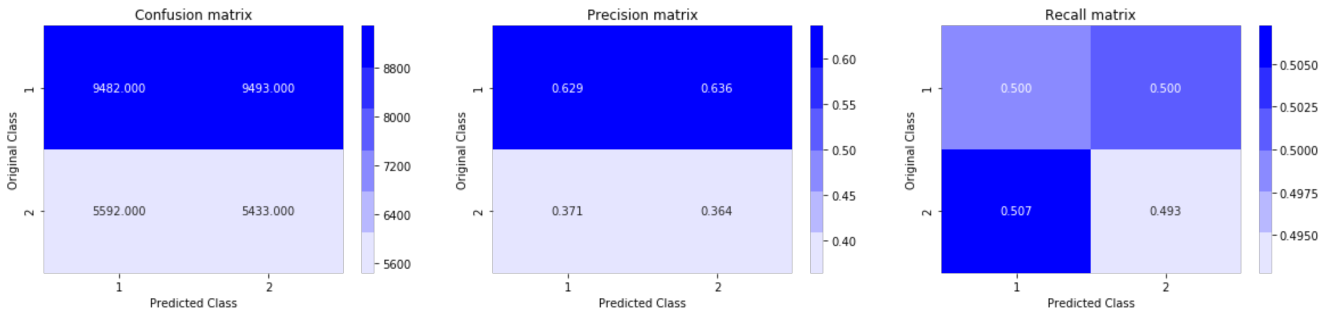
```
    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

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

    plt.show()
```

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

In [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 sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-15))

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

```
Log loss on Test Data using Random Model 0.887242646958
```



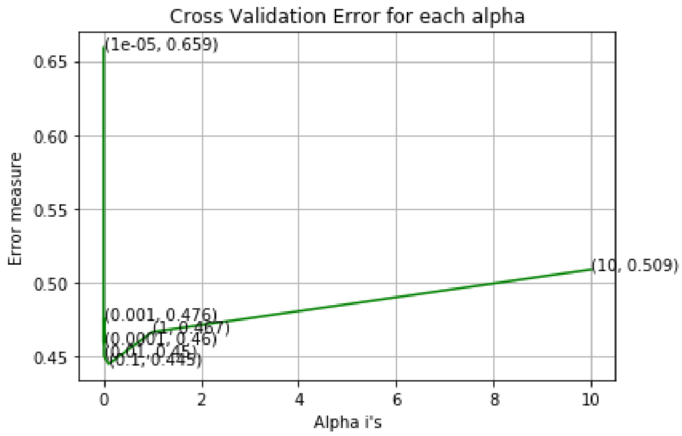
4.4 Logistic Regression with hyperparameter tuning on TFIDF weighted W2V based vectorization

In [0]:

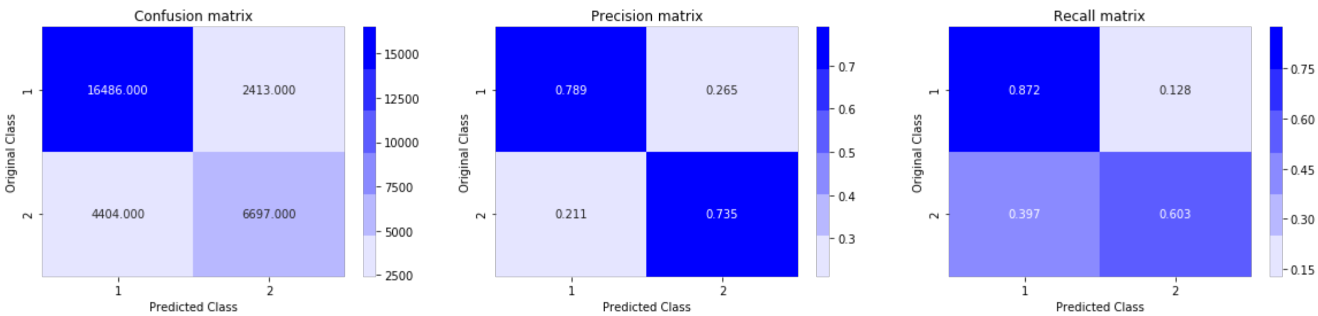
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42 , class_weight='balanced
')
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
```

```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6589734189474593
For values of alpha =  0.0001 The log loss is: 0.4596576080712704
For values of alpha =  0.001 The log loss is: 0.47598914345811755
For values of alpha =  0.01 The log loss is: 0.4504643129769408
For values of alpha =  0.1 The log loss is: 0.44514274763570344
For values of alpha =  1 The log loss is: 0.4668107459743148
For values of alpha =  10 The log loss is: 0.5089490456091396
```




```
For values of best alpha =  0.1 The train log loss is: 0.44207416978281233
For values of best alpha =  0.1 The test log loss is: 0.4502394595299378
Total number of data points : 30000
```



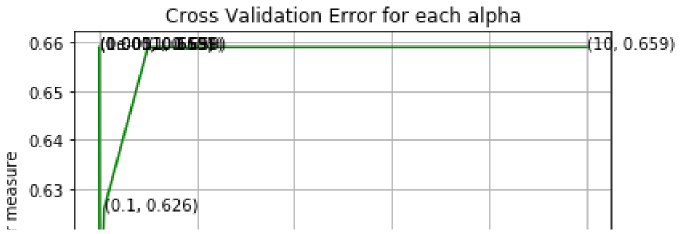
4.5 Linear SVM with hyperparameter tuning on TFIDF weighted W2V based vectorization

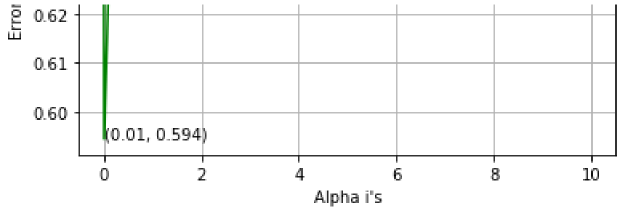
In [0]:

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

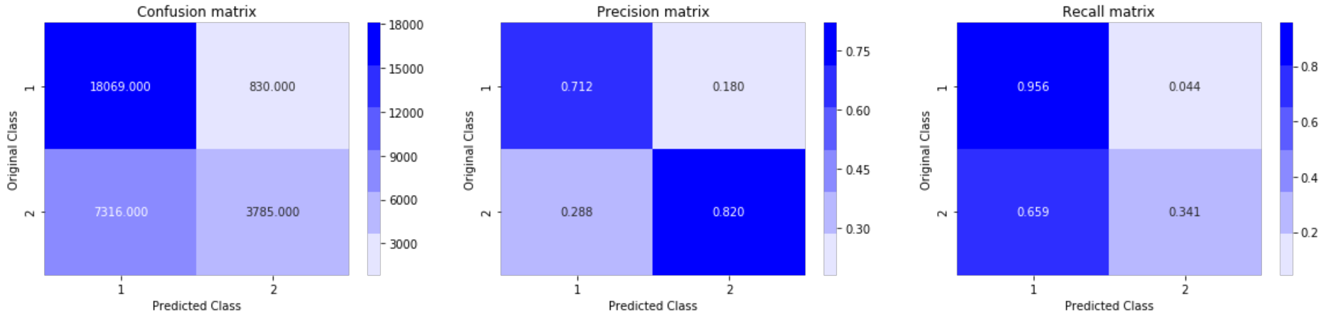
```python
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42 , class_weight='balanc
ed')
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42 , class_we
ight='balanced')
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6589734189474593
For values of alpha =  0.0001 The log loss is: 0.6589734189474593
For values of alpha =  0.001 The log loss is: 0.6589734189474593
For values of alpha =  0.01 The log loss is: 0.5944656394362929
For values of alpha =  0.1 The log loss is: 0.625927058211714
For values of alpha =  1 The log loss is: 0.6589734189474593
For values of alpha =  10 The log loss is: 0.6589734189474593
```






```
For values of best alpha =  0.01 The train log loss is: 0.5449069649156968
For values of best alpha =  0.01 The test log loss is: 0.5457181186452043
Total number of data points : 30000
```



4.6 XGBoost on TFIDF weighted W2V based vectorization

In [0]:

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

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

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

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

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

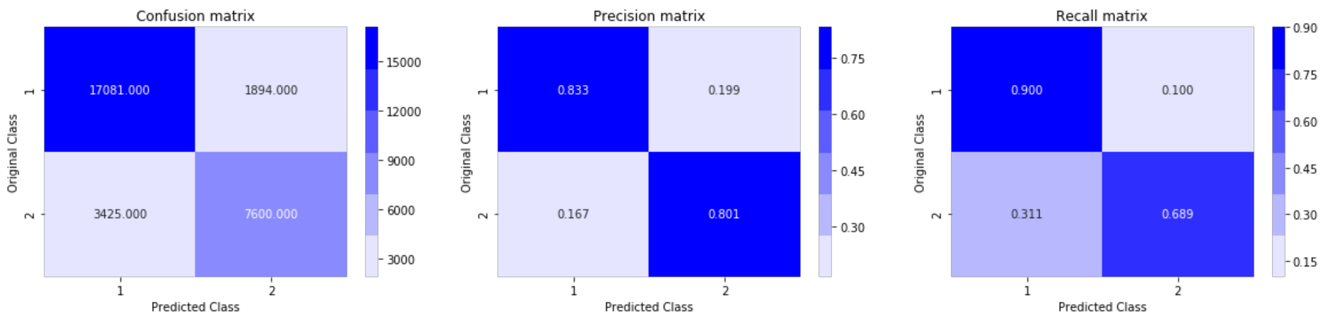
```
[0]	train-logloss:0.684819	valid-logloss:0.684845
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]	train-logloss:0.61583	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

In [12]:

```
df = pd.read_csv(data_path + "nlp_features_train.csv",encoding='latin-1')
df.head(2)
```

Out[12]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	firs
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	0.916659	0.785709	0.0	
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	0.699993	0.466664	0.0	

In [13]:

```
len(df)
df = df.fillna('')
df=df[:100000]
len(df)
```

Out[13]:

100000

In [14]:

```
df_train = df[:70000]
print(len(df_train))
df_test = df[70000:]
print(len(df_test))
```

```
70000
30000
```

In [15]:

```
vectorizer = TfidfVectorizer(min_df=10)
text1_tfidf_train = vectorizer.fit_transform(df_train['question1'])
text1_tfidf_test = vectorizer.transform(df_test['question1'])
print("Shape of matrix after one hot encodig ",text1_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (70000, 5204)
```

In [16]:

```
vectorizer = TfidfVectorizer(min_df=10)
text2_tfidf_train = vectorizer.fit_transform(df_train['question2'])
text2_tfidf_test = vectorizer.transform(df_test['question2'])
print("Shape of matrix after one hot encodig ",text2_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (70000, 5167)
```

In [0]:

```
from scipy.sparse import hstack
```

In [18]:

```
X_train = hstack((np.array(df_train[['id','qid1','qid2','cwc_min','cwc_max','csc_min','csc_max','c
tc_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']]) , text1_tfidf_train , tex
t2_tfidf_train))
X_test =
hstack((np.array(df_test[['id','qid1','qid2','cwc_min','cwc_max','csc_min','csc_max','ctc_min','ct
c_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']]) , text1_tfidf_test ,
text2_tfidf_test))
X1_train=X_train.tocsr()
X1_test=X_test.tocsr()
print(X1_train.shape)
print(X1_test.shape)
```

```
(70000, 10389)
(30000, 10389)
```

In [19]:

```
y_train = df_train['is_duplicate']
y_test = df_test['is_duplicate']
print(y_train.shape)
print(y_test.shape)
```

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

5.1 Logistic Regression with hyperparameter tuning on TFIDF based

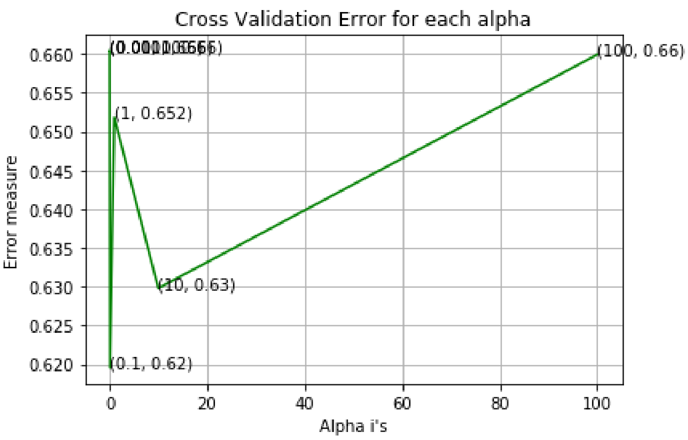
vectorization

5.1.1 With Class weight as Balanced

In [28]:

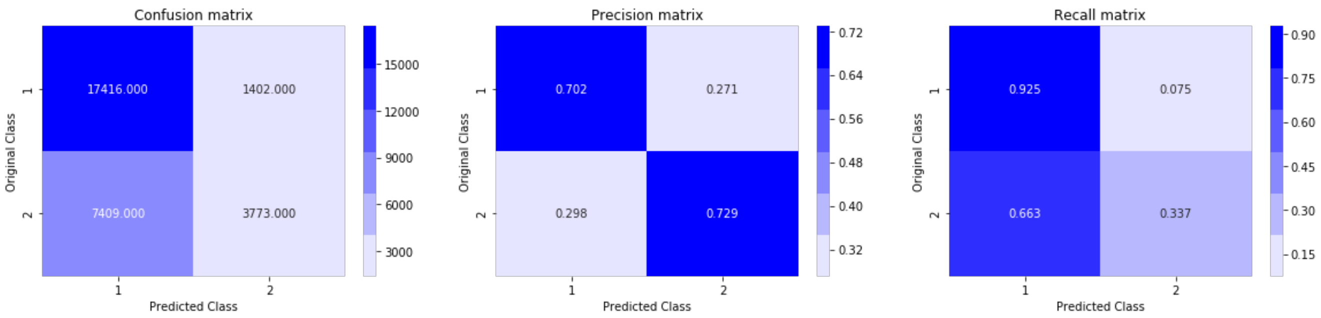
```
alpha = [10 ** x for x in range(-4, 3)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log' , class_weight='balanced')
    clf.fit(X1_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X1_train, y_train)
    predict_y = sig_clf.predict_proba(X1_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log' , class_weight='balanced')
clf.fit(X1_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X1_train, y_train)

predict_y = sig_clf.predict_proba(X1_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X1_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  0.0001 The log loss is: 0.6603945477700923
For values of alpha =  0.001 The log loss is: 0.6603945477700923
For values of alpha =  0.01 The log loss is: 0.6603945477700923
For values of alpha =  0.1 The log loss is: 0.6195181544644561
For values of alpha =  1 The log loss is: 0.6517943474881931
For values of alpha =  10 The log loss is: 0.6297789784714061
For values of alpha =  100 The log loss is: 0.659900447466102
```




```
For values of best alpha =  0.1 The train log loss is: 0.6296961919480639
For values of best alpha =  0.1 The test log loss is: 0.6038942850134336
```

```
Total number of data points : 30000
```

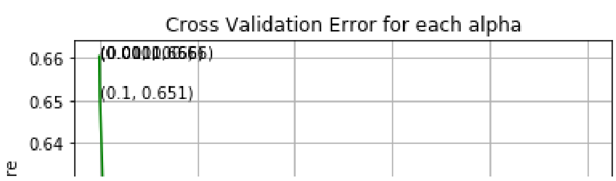


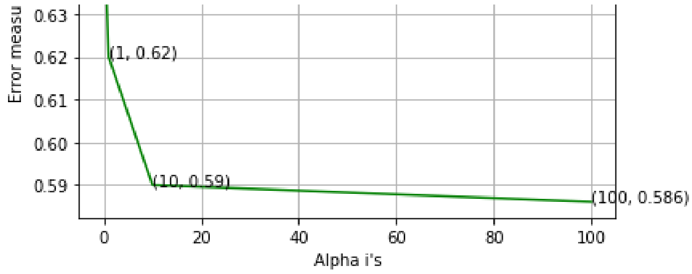
5.1.2 Without Class weight as Balanced

In [29]:

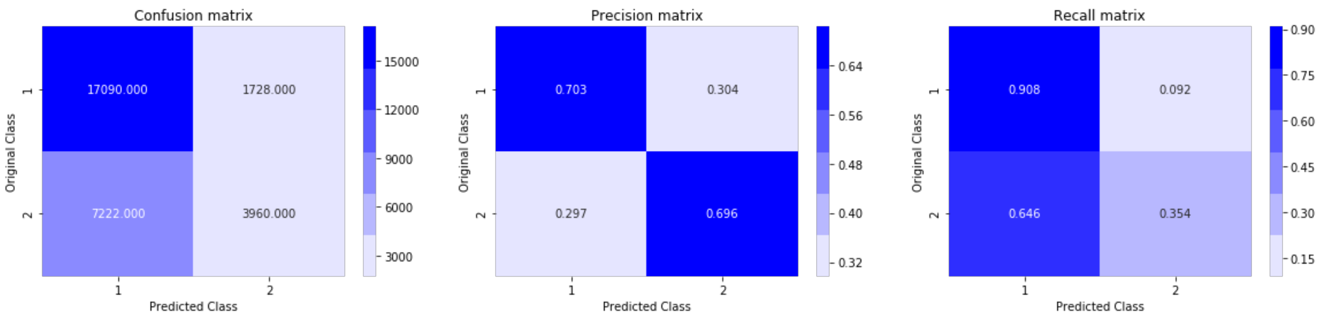
```python
alpha = [10 ** x for x in range(-4, 3)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X1_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X1_train, y_train)
    predict_y = sig_clf.predict_proba(X1_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X1_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X1_train, y_train)

predict_y = sig_clf.predict_proba(X1_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X1_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  0.0001 The log loss is: 0.6603945477700923
For values of alpha =  0.001 The log loss is: 0.6603945477700923
For values of alpha =  0.01 The log loss is: 0.6603945477700923
For values of alpha =  0.1 The log loss is: 0.6509167359319714
For values of alpha =  1 The log loss is: 0.6200529705909165
For values of alpha =  10 The log loss is: 0.5899721593642986
For values of alpha =  100 The log loss is: 0.5860248750661626
```






```
For values of best alpha =  100 The train log loss is: 0.6371126567390029
For values of best alpha =  100 The test log loss is: 0.5860248750661626
Total number of data points : 30000
```



5.2 Linear SVM with hyperparameter tuning on TFIDF based vectorization

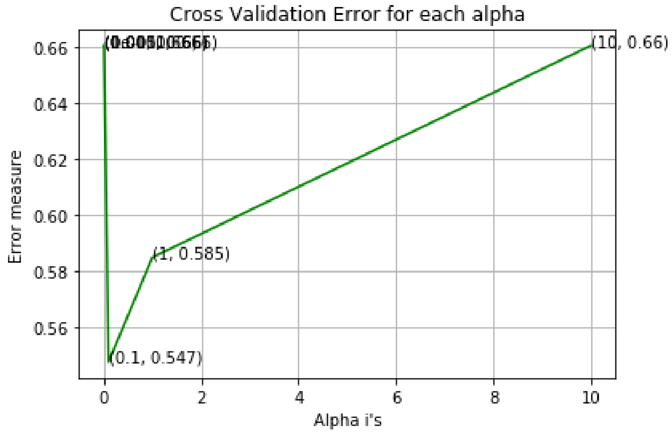
5.1.1 With Class weight as Balanced

In [30]:

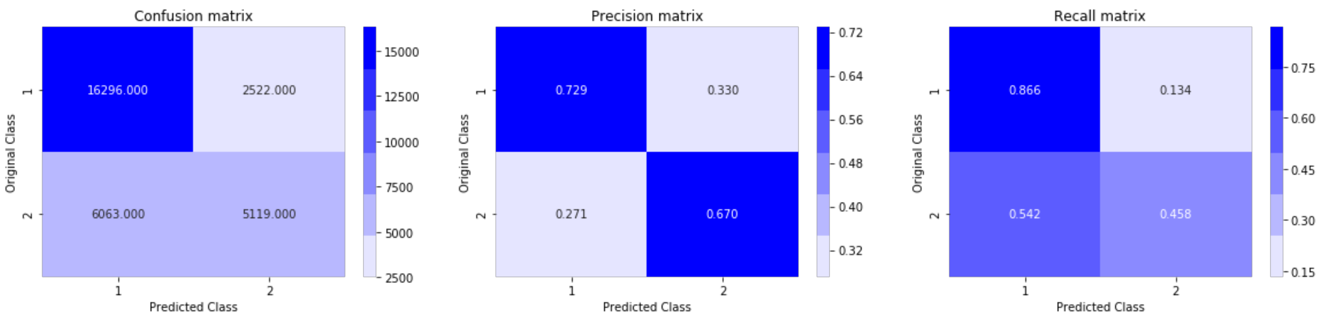
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42 , class_weight='balanc
ed')
    clf.fit(X1_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X1_train, y_train)
    predict_y = sig_clf.predict_proba(X1_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42 , class_we
ight='balanced')
clf.fit(X1_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X1_train, y_train)

predict_y = sig_clf.predict_proba(X1_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X1_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6603945477700923
For values of alpha =  0.0001 The log loss is: 0.6603945477700923
For values of alpha =  0.001 The log loss is: 0.6603945477700923
For values of alpha =  0.01 The log loss is: 0.6603945477700923
For values of alpha =  0.1 The log loss is: 0.5474040222598306
For values of alpha =  1 The log loss is: 0.584768669817024
For values of alpha =  10 The log loss is: 0.6603945477700923
```




```
For values of best alpha =  0.1 The train log loss is: 0.5741202810503914
For values of best alpha =  0.1 The test log loss is: 0.5474040222598306
Total number of data points : 30000
```



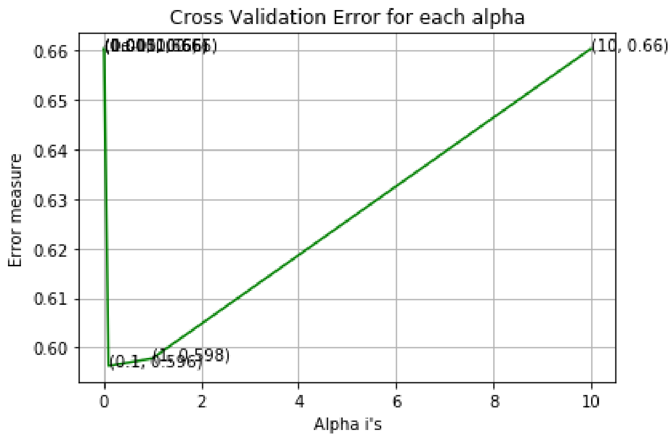
5.1.1 Without Class weight as Balanced

In [31]:

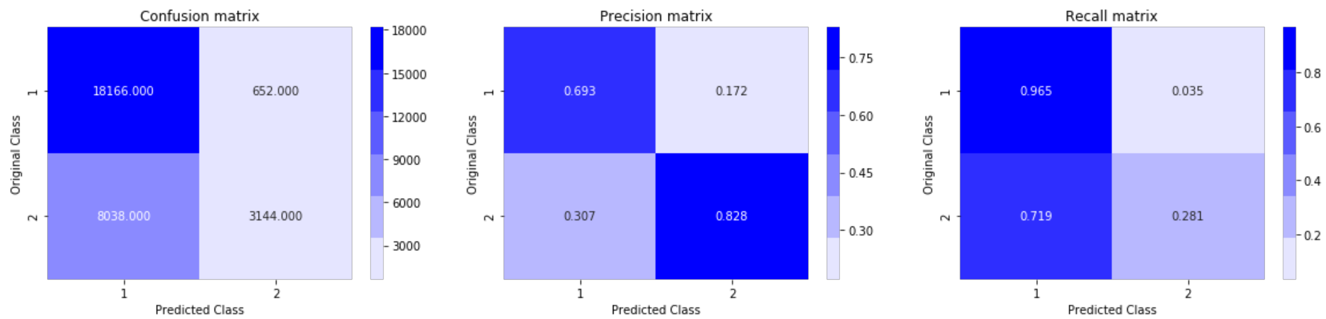
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X1_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X1_train, y_train)
    predict_y = sig_clf.predict_proba(X1_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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

```
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge', random_state=42)
clf.fit(X1_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X1_train, y_train)

predict_y = sig_clf.predict_proba(X1_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X1_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6603945477700923
For values of alpha =  0.0001 The log loss is: 0.6603945477700923
For values of alpha =  0.001 The log loss is: 0.6603945477700923
For values of alpha =  0.01 The log loss is: 0.6603945477700923
For values of alpha =  0.1 The log loss is: 0.5962508713682829
For values of alpha =  1 The log loss is: 0.5977519147303617
For values of alpha =  10 The log loss is: 0.6603945477700923
```




```
For values of best alpha =  0.1 The train log loss is: 0.6266487755781842
For values of best alpha =  0.1 The test log loss is: 0.5962508713682829
Total number of data points : 30000
```



5.3 XGBoost with Hyperparameter Tuning on TFIDF based vectorization

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
from datetime import datetime
xgb = XGBClassifier()
clf = RandomizedSearchCV(xgb, params, cv=5)
start_time = timer(None) # timing starts from this point for "start_time" variable
clf.fit(X1_train, y_train)
timer(start_time) # timing ends here for "start_time" variable
```

```
Time taken: 0 hours 51 minutes and 24.76 seconds.
```

In [0]:

```
clf.best_params_
```

```
{'learning_rate': 0.1,
 'max_depth': 7,}
```

In [0]:

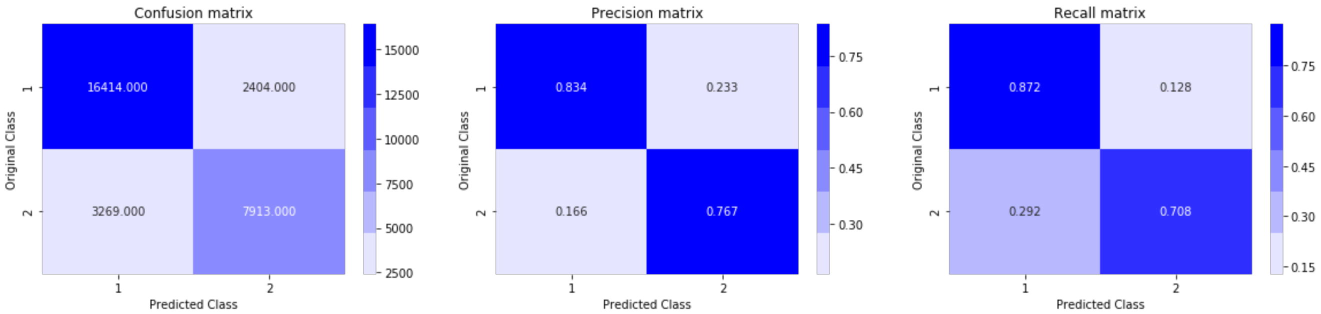
```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.1
params['max_depth'] = 7

d_train = xgb.DMatrix(X1_train, label=y_train)
d_test = xgb.DMatrix(X1_test, label=y_test)

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

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

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

```
[0] train-logloss:0.656874 valid-logloss:0.656017
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10] train-logloss:0.493699 valid-logloss:0.492732
[20] train-logloss:0.441675 valid-logloss:0.445827
[30] train-logloss:0.419565 valid-logloss:0.42762
[40] train-logloss:0.406935 valid-logloss:0.417782
[50] train-logloss:0.39803 valid-logloss:0.41141
[60] train-logloss:0.390751 valid-logloss:0.40802
[70] train-logloss:0.385838 valid-logloss:0.405492
[80] train-logloss:0.380294 valid-logloss:0.40285
[90] train-logloss:0.376155 valid-logloss:0.400955
[100] train-logloss:0.372537 valid-logloss:0.398314
[110] train-logloss:0.36878 valid-logloss:0.396632
[120] train-logloss:0.365682 valid-logloss:0.395615
[130] train-logloss:0.362759 valid-logloss:0.394406
[140] train-logloss:0.360011 valid-logloss:0.3935
[150] train-logloss:0.357383 valid-logloss:0.392486
[160] train-logloss:0.354643 valid-logloss:0.391496
[170] train-logloss:0.351753 valid-logloss:0.389005
[180] train-logloss:0.349142 valid-logloss:0.388219
[190] train-logloss:0.347176 valid-logloss:0.387547
[200] train-logloss:0.345148 valid-logloss:0.386932
[210] train-logloss:0.343052 valid-logloss:0.386377
[220] train-logloss:0.34147 valid-logloss:0.385705
[230] train-logloss:0.339635 valid-logloss:0.385283
[240] train-logloss:0.337196 valid-logloss:0.383486
[250] train-logloss:0.334793 valid-logloss:0.382723
[260] train-logloss:0.333202 valid-logloss:0.382337
[270] train-logloss:0.331062 valid-logloss:0.381668
[280] train-logloss:0.329744 valid-logloss:0.381325
[290] train-logloss:0.328071 valid-logloss:0.380876
[300] train-logloss:0.326183 valid-logloss:0.380512
[310] train-logloss:0.324586 valid-logloss:0.380007
[320] train-logloss:0.322927 valid-logloss:0.379676
[330] train-logloss:0.321289 valid-logloss:0.379057
[340] train-logloss:0.320157 valid-logloss:0.378755
[350] train-logloss:0.318757 valid-logloss:0.378562
[360] train-logloss:0.317639 valid-logloss:0.378362
[370] train-logloss:0.316354 valid-logloss:0.378032
[380] train-logloss:0.315156 valid-logloss:0.377823
[390] train-logloss:0.313446 valid-logloss:0.377353
[399] train-logloss:0.312387 valid-logloss:0.377094
The test log loss is: 0.3770938920243238
```

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
```



6. Conclusion

In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
x = PrettyTable()
x.field_names = ["Algorithm", "Vectorizer", "Model", "Hyper Parameters", "Log-Loss"]
```

In [0]:

```
x.add_row(["Logistic Regression + class balancing", "Tfidf W2V", "Brute","alpha=0.1", 0.450])
x.add_row(["Linear SVM + class balancing", "Tfidf W2V", "Brute","alpha=0.01", 0.545])
x.add_row(["XGBoost", "Tfidf W2V", "Brute","eta=0.02 , max_depth=4", 0.357])
x.add_row(["Logistic Regression", "Tfidf", "Brute","alpha=100", 0.586])
x.add_row(["Linear SVM", "Tfidf", "Brute","alpha=0.1", 0.596])
x.add_row(["Logistic Regression + class balancing", "Tfidf", "Brute","alpha=0.1", 0.603])
x.add_row(["Linear SVM + class balancing", "Tfidf", "Brute","alpha=0.1", 0.547])
x.add_row(["XGBoost", "Tfidf", "Brute","eta=0.1 , max_depth=7", 0.377])
```

In [43]:

```
print(x)
```

```
+---------------------------------------+------------+-------+-------------------------+----------+
|               Algorithm               | Vectorizer | Model |     Hyper Parameters    | Log-Loss |
+---------------------------------------+------------+-------+-------------------------+----------+
| Logistic Regression + class balancing | Tfidf W2V  | Brute |        alpha=0.1        |   0.45   |
|      Linear SVM + class balancing     | Tfidf W2V  | Brute |        alpha=0.01       |  0.545   |
|                XGBoost                | Tfidf W2V  | Brute | eta=0.02 , max_depth=4  |  0.357   |
|          Logistic Regression          |   Tfidf    | Brute |        alpha=100        |  0.586   |
|               Linear SVM              |   Tfidf    | Brute |        alpha=0.1        |  0.596   |
| Logistic Regression + class balancing |   Tfidf    | Brute |        alpha=0.1        |  0.603   |
|      Linear SVM + class balancing     |   Tfidf    | Brute |        alpha=0.1        |  0.547   |
|                XGBoost                |   Tfidf    | Brute | eta=0.1 , max_depth=7   |  0.377   |
+---------------------------------------+------------+-------+-------------------------+----------+
```

Steps that were followed during this case study

1. Initially to understand the data better Exploratory data analysis was performed and the results were taken into account.
2. There was a need for feature engineering in this task therefore after going through various blogs regarding this problem and discussions that happened on kaggle certain important and very elegant features were added to the dataset to make it information rich.
3. The new features are then analysed to see if they are actually important and do they really help in classification of data.
4. All the data is then compiled and models are applied on top of it. The data features are taken in two ways - TFIDF and TFIDF

weighted W2V.

5. Three models are implied on top of it - Linear SVM, Logistic Regression, XGBoost.
6. The Logerithmic loss for all the three models for different vectorizers is compiled in the above table which clearly shows that XGBoost with TFIDF W2V vectorizer performs the best(considering the log-loss metric) with just 0.357 log loss.